Triple Data Encryption Standard

Project Report submitted in partial fulfillment

of the requirement for the degree of

Bachelor of Technology

in

Computer Science & Engineering

under the Supervision of

Mr. Amit Kumar Singh

By

Mudit Singh (*111291*)

to

Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

Certificate

This is to certify that project report entitled "**Implementation of Triple Data Encryption Standards (3-DES)**", submitted by **Mudit Singh** in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science & Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Amit Kumar Singh

Assistant Professor

Dept. of CSE

Date :

Acknowledgement

It is my pleasure to be indebted to various people, who directly or indirectly contributed in the development of this work and who influenced my thinking, behavior, and acts during the course of our training.

I express my sincere gratitude to **Mr. Amit Kumar Singh,** worthy mentor, guide and a great teacher who influenced and inspired me in many ways and for introducing the present topic and for their inspiring guidance, constructive criticism and valuable suggestion throughout the project work.

I would also like to thank **Prof. Dr. Satya Prakash Ghrera** for sharing his vast expanse of knowledge in guiding me with the correct books and sparing his valuable time and helped me in striving to move forward to this point. Without their valuable inputs, I wouldn't have been able to incrementally work well and go ahead with the project.

Lastly, I would like to thank my friends with whom I shared my day-to-day experience and received lots of suggestions that improved my quality of work.

Date: **Mudit Singh**

Table of Content

List of Figures

List of Tables

Abstract

The future of the Internet, especially in expanding the range of applications, involves a much deeper degree of privacy, and authentication. Without these the Internet cannot be properly used to replace existing applications such as in voting, finance, and so on.

Many people are not aware that the information they send or the files stored on their computers needs to be protected, however when you consider what you have on your computer and the many ways it can fall into the wrong hands, it does start to make sense to protect your privacy in some way.

The future is thus towards data encryption which is the science of cryptography , and provides a mechanism for two entities to communicate without any other entity being able to read their messages.

This report illustrates how to encrypt a text and to decrypt the encrypted form of the text to get the original input text. This encryption and decryption is done using a derived form of the most widely used private key encryption scheme based on data encryption standard, Triple DES (3-DES).

Chapter 1

3-DES: AN OVERVIEW

The future of the Internet, especially in expanding the range of applications, involves a much deeper degree of privacy, and authentication. Without these the Internet cannot be properly used to replace existing applications such as in voting, finance, and so on.

The future is thus towards data encryption which is the science of cryptography , and provides a mechanism for two entities to communicate without any other entity being able to read their messages.

1.1 Purpose

If you don't all ready know it, without encryption there is no such thing as privacy. At least not for your data. It's all 1's and 0's but doesn't take a genius at all to recognize the data it represents if it's not encrypted when intercepted. And there are literally thousands of ways to intercept data. So to protect your privacy, some encryption schemes were developed.

One of the such method is Data Encryption Standard. The DES algorithm has been around for a long time, and the 56-bit version is now easily crack able (in less than a day on fairly modest equipment). An enhancement, and one which is still fairly compatible with DES, is the 3-DES algorithm.

1.2 Motivation

When it was discovered that the key size of the original DES cipher was becoming subject to brute force attacks because of the availability of increasing computational power, Triple DES was designed to provide a relatively simple method of increasing the key size of DES to protect against such attacks, without designing a completely new block cipher algorithm. The use of three steps is essential to prevent meet-in-the-middle attacks that are effective against double DES encryption.

1.3 Data Encryption Standard

Data Encryption Standard (DES) is a symmetric block cipher developed by IBM [1]. The algorithm uses a 56-bit key to encipher/decipher a 64-bit block of data. The key is always presented as a 64-bit block, every 8th bit of which is ignored. However, it is usual to set each 8th bit so that each group of 8 bits has an odd number of bits set to 1.

The algorithm is best suited to implementation in hardware, probably to discourage implementations in software, which tend to be slow by comparison. However, modern computers are so fast that satisfactory software implementations are readily available.

DES is the most widely used symmetric algorithm in the world, despite claims that the key length is too short. Ever since DES was first announced, controversy has raged about whether 56 bits is long enough to guarantee security.

The key length argument goes like this. Assuming that the only feasible attack on DES is to try each key in turn until the right one is found, then 1,000,000 machines each capable of testing

1,000,000 keys per second would find (on average) one key every 12 hours. Most reasonable people might find this rather comforting and a good measure of the strength of the algorithm.

Those who consider the exhaustive key-search attack to be a real possibility (and to be fair the technology to do such a search is becoming a reality) can overcome the problem by using double or triple length keys. In fact, double length keys have been recommended for the financial industry for many years.

And despite the recent coup by the Electronic Frontier Foundation in creating a $220,000 machine to crack DES-encrypted messages, DES will live on in government and banking for years to come through a life- extending version called Triple Data Encryption Standard (3DES).

1.4 Triple-Data Encryption Standard[1]

Use of multiple length keys leads us to the Triple-DES algorithm, in which DES is applied three times. Triple DES is simply another mode of DES operation. It takes three 64-bit keys, for an overall key length of 192 bits. In Private Encryption, you simply type in the entire 192-bit (24 character) key rather than entering each of the three keys individually.

The Triple DES DLL then breaks the user provided key into three sub keys, padding the keys if necessary so they are each 64 bits long. The procedure for encryption is exactly the same as regular DES, but it is repeated three times. Hence the name Triple DES, The data is encrypted with the first key, decrypted with the second key, and finally encrypted again with the third key.

Consequently, Triple DES runs three times slower than standard DES, but is much more secure if used properly. The procedure for decrypting something is the same as the procedure for encryption, except it is executed in reverse. Like DES, data is encrypted and decrypted in 64-bit chunks. Unfortunately, there are some weak keys that one should be aware of: if all three keys, the first and second keys, or the second and third keys are the same, then the encryption procedure is essentially the same as standard DES. This situation is to be avoided because it is the same as using a really slow version of regular DES.

Although the input key for DES is 64 bits long, the actual key used by DES is only 56 bits in length. The least significant (right-most) bit in each byte is a parity bit, and should be set so that there are always an odd number of 1s in every byte. These parity bits are ignored, so only the seven most significant bits of each byte are used, resulting in a key length of 56 bits. This means that the effective key strength for Triple DES is actually 168 bits because each of the three keys contains 8 parity bits that are not used during the encryption process.

If we consider a triple length key to consist of three 56-bit keys K1, K2, K3 then encryption is as follows:

•Encrypt with K1 •Decrypt with K2 •Encrypt with K3

Decryption is the reverse process:

•Decrypt with K3 •Encrypt with K2 •Decrypt with K1

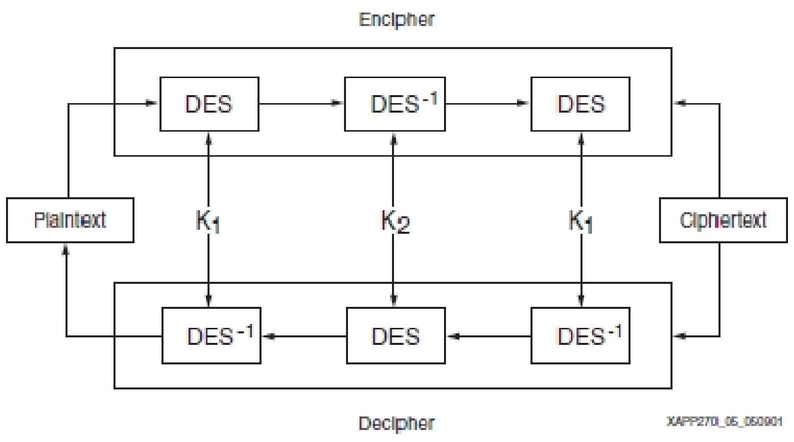


Figure 1.1. Triple DES Block Diagram

- Setting K3 equal to K1 in these processes gives us a double length key K1, K2.

- Setting K1, K2 and K3 all equal to K has the same effect as using a single-length (56-bit key). Thus it is possible for a system using triple-DES to be compatible with a system using single-DES.

DES operates on a 64 – bit block of plaintext. After an initial permutation the block is broken into a right half and left half, each 32 – bits long. Then there are 16 rounds of identical operations, called Function f, in which the data are combined with the key. After the sixteenth round, the right and left halves are joined, and a final permutation (the inverse of the initial permutation) finishes off the algorithm.

In each round the key bits are shifted, and then 48 – bits are selected from the 56 –bits of the key. The right half of the data is expanded to 48 – bits via an expansion permutation, combined with 48 –bits of a shifted and permuted key via an XOR, sent through 8 S- boxes producing 32-new bits, and permuted again. These four operations make up Function f. The output of

Function f is then combined with the left half via another XOR. The results of these operations become the new right half; the old right half becomes the new left half. These operations are repeated sixteen times, making 16 rounds of DES.

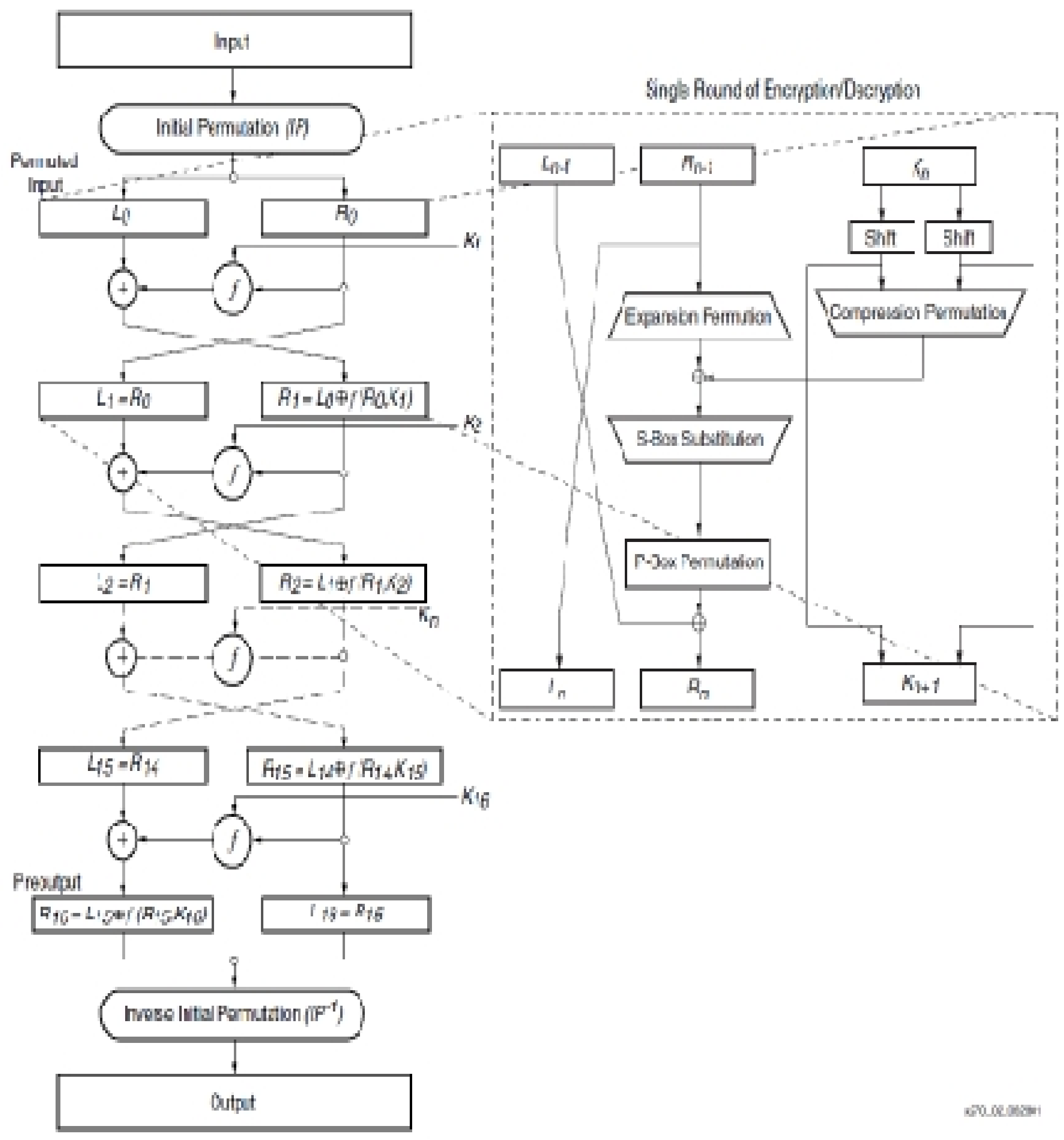


Figure 1.2. DES Algorithm

1.5 Background [2]

Table 1.1. Chronology of 3DES

Date	Year	Event
15 May	1973	NBS publishes a first request for a standard encryption algorithm
27 August	1974	NBS publishes a second request for encryption algorithms
17 March	1975	DES is published in the *Federal Register* for comment
August	1976	First workshop on DES
September	1976	Second workshop, discussing mathematical foundation of DES
November	1976	DES is approved as a standard

15 January	1977	DES is published as a FIPS standard FIPS PUB 46
	1983	DES is reaffirmed for the first time
	1986	Video cipher II, a TV satellite scrambling system based upon DES, begins use by HBO
22 January	1988	DES is reaffirmed for the second time as FIPS 46-1, superseding FIPS PUB 46
July	1991	Biham and Shamir rediscover differential cryptanalysis, and apply it to a 15-round DES-like cryptosystem.
	1992	Biham and Shamir report the first theoretical attack with less complexity than brute force: differential cryptanalysis. However, it requires an unrealistic 2^{47} chosen plaintexts.
30 Dec	1993	DES is reaffirmed for the third time as FIPS 46-2
	1994	The first experimental cryptanalysis of DES is performed using linear cryptanalysis (Matsui, 1994).
June	1997	The DESCHALL Project breaks a message encrypted with DES for the first time in public.
July	1998	The EFF's DES cracker (Deep Crack) breaks a DES key in 56 hours.

January	1999	Together, Deep Crack and distributed.net break a DES key in 22 hours and 15 minutes.
25 October	1999	DES is reaffirmed for the fourth time as FIPS 46-3, which specifies the preferred use of 3DES.

On May 15, 1973, during the reign of Richard Nixon, the National Bureau of Standards (NBS) published a notice in the Federal Register soliciting proposals for cryptographic algorithms to protect data during transmission and storage. The notice explained why encryption was an important issue.

Over the last decade, there has been an accelerating increase in the accumulations and communication of digital data by government, industry and by other organizations in the private sector. The contents of these communicated and stored data often have very significant value and/or sensitivity. It is now common to find data transmissions which constitute funds transfers of several million dollars, purchase or sale of securities, warrants for arrests or arrest and conviction records being communicated between law enforcement agencies, airline reservations and ticketing representing investment and value both to the airline and passengers, and health and patient care records transmitted among physicians and treatment centers.

The increasing volume, value and confidentiality of these records regularly transmitted and stored by commercial and government agencies has led to heightened recognition and concern over their exposures to unauthorized access and use. This misuse can be in the form of theft or defalcations of data records representing money, malicious modification of business inventories or the interception and misuse of confidential information about people. The need for protection is then apparent and urgent.

It is recognized that encryption (otherwise known as scrambling, enciphering or privacy transformation) represents the only means of protecting such data during transmission and a useful means of protecting the content of data stored on various media, providing encryption of adequate strength can be devised and validated and is inherently integrable into system architecture. The National Bureau of Standards solicits proposed techniques and algorithms for computer data encryption. The Bureau also solicits recommended techniques for implementing the cryptographic function: for generating, evaluating, and protecting cryptographic keys; for maintaining files encoded under expiring keys; for making partial updates to encrypted files; and mixed clear and encrypted data to permit labeling, polling, routing, etc. The Bureau in its role for establishing standards and aiding government and industry in assessing technology, will arrange for the evaluation of protection methods in order to prepare guidelines.

NBS waited for the responses to come in. It received none until August 6, 1974, three days before Nixon's resignation, when IBM submitted a candidate that it had developed internally under the name LUCIFER. After evaluating the algorithm with the help of the National Security Agency (NSA), the NBS adopted a modification of the LUCIFER algorithm as the *new Data Encryption Standard (DES) on July 15, 1977.*

In May 2005, DES was withdrawn, and is now only approved as a component of TDEA. You will still encounter it on occasion, such as with a Microsoft VPN using PPTP.

Triple-DES was specified by NIST in May 2004 by SP 800-67.

1.6 Definitions [3]

- ***Cryptanalysis*** is the study of mathematical techniques for attempting to defeat cryptographic techniques, and, more generally, information security services.
- A ***cryptanalysts*** is someone who engages in cryptanalysis.
- ***Cryptology*** is the study of cryptography and cryptanalysis.
- ***Cryptosystem*** is a general term referring to a set of cryptography primitives used to provide information security services. Most often the term is used in conjunction with primitives providing confidentiality, i.e. Encryption.
- ***Plaintext***: This is the original intelligible message or data that is fed into the algorithm as input.

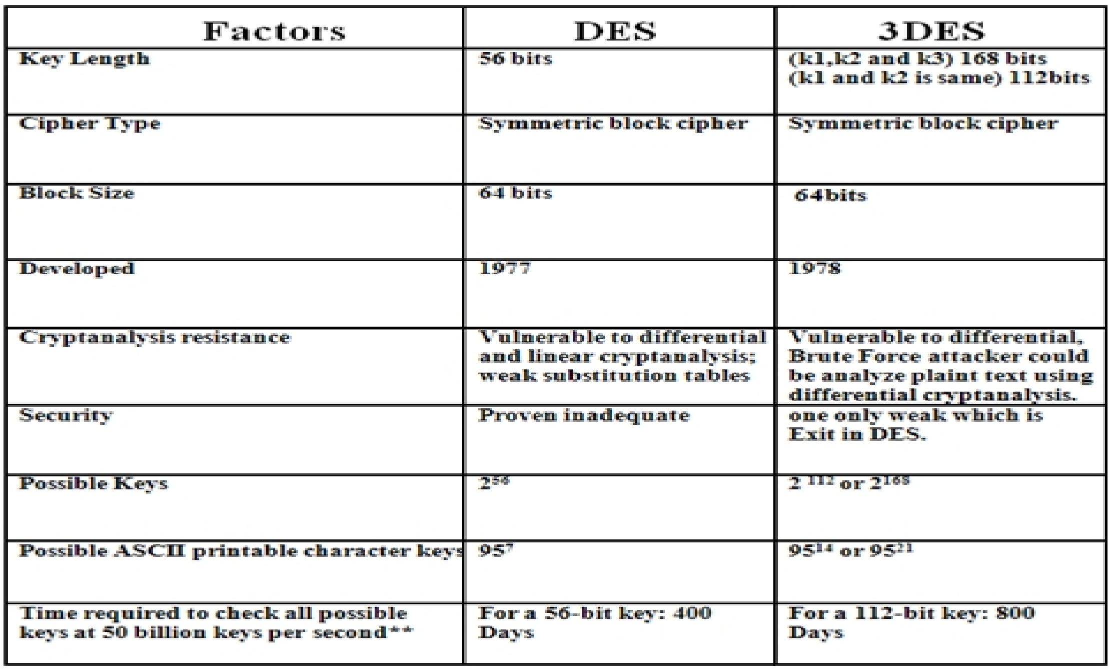
- ***Encryption algorithm***: The encryption algorithm performs various substitutions and transformations on the plaintext.
- ***Decryption algorithm*:** This is essentially the encryption algorithm run is reverse. It takes the cipher text and the secret keys and produces the original plaintext.
- ***Secret key***: The secret key is also input to the encryption algorithm. The key is a value independent of the plaintext. The algorithm will produce a different output depending on the specific key being used at the time.
- ***Cipher text*:** This is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different cipher text. The cipher text is an apparently random stream of data and, as it stands, is unintelligible.

1.7 Comparison between 3DES & DES [4]

DES is the old "data encryption standard" from the seventies. Its key size is too short for proper security (56 effective bits; this can be brute-forced, as has been demonstrated more than ten years ago). Also, DES uses 64-bit blocks, which raises some potential issues when encrypting several gigabytes of data with the same key.

3DES is a trick to reuse DES implementations, by cascading three instances of DES (with distinct keys). 3DES is believed to be secure up to at least "2^{112}" security (which is quite a lot, and quite far in the realm of "not breakable with today's technology"). But it is slow, especially in software (DES was designed for efficient hardware implementation, but it sucks in software; and 3DES sucks three times as much).

Table 1.2. Triple DES vs DES



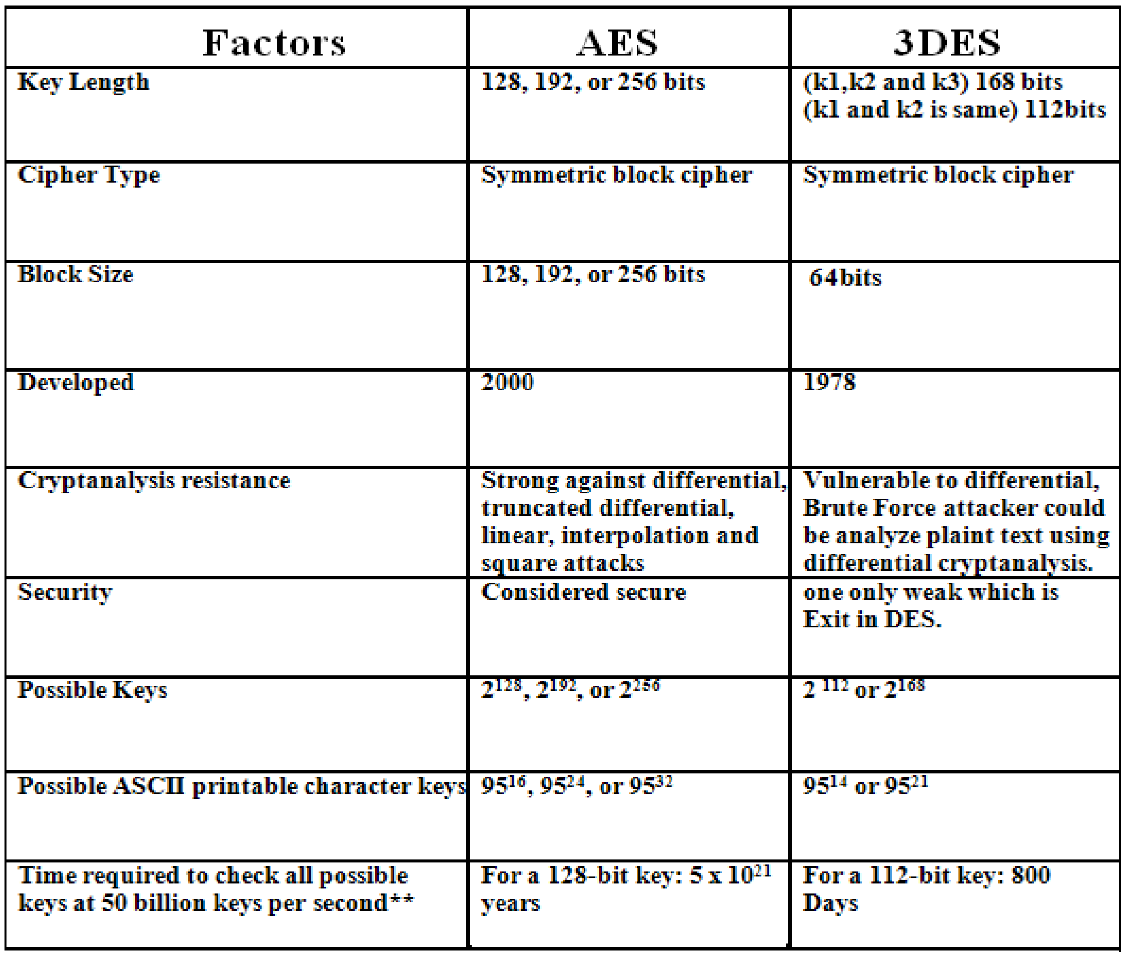
Factors	DES	3DES
Key Length	56 bits	(k1,k2 and k3) 168 bits (k1 and k2 is same) 112bits
Cipher Type	Symmetric block cipher	Symmetric block cipher
Block Size	64 bits	64bits
Developed	1977	1978
Cryptanalysis resistance	Vulnerable to differential and linear cryptanalysis; weak substitution tables	Vulnerable to differential, Brute Force attacker could be analyze plaint text using differential cryptanalysis.
Security	Proven inadequate	one only weak which is Exit in DES.
Possible Keys	2^{56}	2^{112} or 2^{168}
Possible ASCII printable character keys	95^{7}	95^{14} or 95^{21}
Time required to check all possible keys at 50 billion keys per second**	For a 56-bit key: 400 Days	For a 112-bit key: 800 Days

1.8 Comparison between 3DES & AES [4]

AES is the successor of DES as standard symmetric encryption algorithm for US federal organizations. AES uses keys of 128, 192 or 256 bits, although, 128 bit keys provide sufficient strength today. It uses 128 bit blocks, and is efficient in both software and hardware implementations. It was selected through an open competition involving hundreds of cryptographers during several years.

3DES is a way to reuse DES implementations, by chaining three instances of DES with different keys. 3DES is believed to still be secure because it requires 2^112 operations which is not achievable with foreseeable technology. 3DES is very slow especially in software implementations because DES was designed for performance in hardware.

Table 1.3. Triple DES vs AES



Factors	AES	3DES
Key Length	128, 192, or 256 bits	(k1,k2 and k3) 168 bits (k1 and k2 is same) 112bits
Cipher Type	Symmetric block cipher	Symmetric block cipher
Block Size	128, 192, or 256 bits	64bits
Developed	2000	1978
Cryptanalysis resistance	Strong against differential, truncated differential, linear, interpolation and square attacks	Vulnerable to differential, Brute Force attacker could be analyze plaint text using differential cryptanalysis.
Security	Considered secure	one only weak which is Exit in DES.
Possible Keys	2^{128}, 2^{192}, or 2^{256}	2^{112} or 2^{168}
Possible ASCII printable character keys	95^{16}, 95^{24}, or 95^{32}	95^{14} or 95^{21}
Time required to check all possible keys at 50 billion keys per second**	For a 128-bit key: 5 x 10^{21} years	For a 112-bit key: 800 Days

1.9 Cryptanalysis [5]

Cryptanalysis refers to the study of ciphers, cipher text, or cryptosystems (that is, to secret code systems) with a view to finding weaknesses in them that will permit retrieval of the plaintext from the cipher text, without necessarily knowing the key or the algorithm. This is known as *breaking* the cipher, cipher text, or cryptosystem.

There are two categories of cryptography.

1.Symmetric key cryptography

2.Asymmetric key cryptography

In symmetric key, there is only single key which is used by sender for encryption and receiver for decryption. In this type the key is shared between both the parties[4].In asymmetric key, there are two keys: a private key and a public key. Private key is kept by receiver for decryption and public key is announced to public and used for encrypt the data. Now various new techniques are developing for encryption of data as well as various techniques are also evolving in order to hack that data.

Breaking is sometimes used interchangeably with *weakening*. This refers to finding a property (fault) in the design or implementation of the cipher that reduces the number of keys required in a brute force attack (that is, simply trying every possible key until the correct one is found). For example, assume that a symmetric cipher implementation uses a key length of 2^{128} bits: this means that a brute force attack would need to try up to all 2^{128} possible combinations (rounds) to be certain of finding the correct key (or, on average, 2^{127} possible combinations) to convert the cipher text into plaintext, which is not possible given present and near future computing abilities. However, a cryptanalysis of the cipher reveals a technique that would allow the plaintext to be found in 2^{40} rounds. While not completely broken, the cipher is now much weaker and the plaintext can be found with moderate computing resources.

1.10 Classification Of Attacks [3,5]

Attacks can be classified based on what type of information the attacker has available. As a basic starting point it is normally assumed that, for the purposes of analysis, the general algorithm is known; this is Shannon's Maxim "the enemy knows the system"—in its turn, equivalent to Kerckhoffs' principle. This is a reasonable assumption in practice — throughout history, there are countless examples of secret algorithms falling into wider knowledge, variously through espionage, betrayal and reverse engineering. (And on occasion, ciphers have been reconstructed through pure deduction; for example, the German Lorenz cipher and the Japanese Purple code, and a variety of classical schemes).

Classification of attacks can be done on following basis:

1.10.1. Amount of Information Available to Attacker

Cipher text only attack : The enemy has intercepted cipher text but has no matching plaintext.

You typically assume that the enemy has access to the cipher text. Two situations:

a) The enemy is aware of the nature of the cryptosystem, but does not have the key. True

with most cryptosystems used in U.S. businesses.

b) The enemy is not aware of the nature of the cryptosystem. The proper users should never assume that this situation will last very long. The Skipjack algorithm on the Clipper Chip is classified, for example. Often the nature of a military cryptosystem is kept secret as long as possible. RSA has tried to keep the nature of a few of its cryptosystems secret, but they were published on Cypherpunks.

Known plaintext attack **:** The enemy has some matched cipher text/plaintext pairs. The

enemy may well have more cipher text also.

Chosen plaintext attack **:** Here we assume that the enemy can choose the plaintext that he wants put through the cryptosystem. Though this is, in general, unrealistic, such attacks are of theoretic interest because if enough plaintext is known, then chosen plaintext attack techniques may be useable. However this is an issue with smart cards.

Chosen Cipher text : The attacker obtain the various plaintext corresponding to an arbitrary set of cipher text.

Adaptive Chosen Plaintext **:** This is similar with the Chosen Plaintext, except in this attacker chooses subsequent set of plaintext which is based on the information obtain from previous encryption methods.

Adaptive Chosen Cipher text :This is similar with the Chosen Cipher text, except in this attacker chooses subsequent set of cipher text which is based on the information obtain from previous encryption methods(previous results).

Related Key Attack**:** Like the chosen plaintext, attack in which attacker can obtain only cipher text encrypted with the help of two keys. These keys are unknown but the relationship between these keys is known. example, two keys differ by a single bit.

1.10.2. On the Basis of Computational Resources Required

Attacks can also be characterized by the resources they require. Those resources include:

Time **:** the number of computation steps (e.g., test encryptions) which must be performed.

Memory : the amount of storage required to perform the attack.

Data : the quantity and type of plaintexts and cipher texts required for a particular approach.

It's sometimes difficult to predict these quantities precisely, especially when the attack isn't practical to actually implement for testing. But academic cryptanalysts tend to provide at least the estimated order of magnitude of their attacks' difficulty.

1.10.3. On the Basis of Partial Breaks

The result of cryptanalysis also varies in terms of usefulness of that data. Cryptographer Lars Knudsen classified various types of attacks on the basis of amount and quality of secret information that is discovered.

Total Break: In this attacker find out the secret key.

Global Deduction: In this attacker find out equivalent algorithm for encryption and decryption without knowing secret key.

Instance Deduction: Attacker find out some additional cipher text or plaintext without previously known.

Information Deduction: Attacker find out some Shannon information about plaintext or cipher text not previously known.

Distinguishing Algorithm: Attacker differentiate various cipher text from random permutation.

In academic cryptography, it is very difficult to specify break or weakness so it is define quite conservatively. It usually require unknown or impractical amount of time, money and cipher text .The attacker may be able to do various things the real world attacker can’t do. Furthermore, it might only reveal a very small information in order to specify that cryptosystem is imperfect but this information is not that much useful for attacker. Finally attacker may attack only on the weakened version of cryptographic tools, like a reduced robin block cipher, as a step towards break the full system.

1.11 Cryptanalysis Of Symmetric Cipher [6]

There are various types of attacks done on symmetric cipher. The explanation is given below:

1.11.1. Boomerang Attack

This is a method of cryptanalysis of block cipher based on differential cryptanalysis. This attack provide various avenues of attack on various cipher which are deemed safe from differential cryptanalysis.

This attack is used to generate so called "quartet" at the point halfway through the cipher. For this purpose an encryption action E is split into its two consecutive stages E0 and E1 so that E(M)=E1(E0(M)),where M is plaintext message.

1.11.2. Brute Force Attack

Brute force attack or exhaustive key search is a type of strategy which can be applied on any type of encrypted data. In this type of attack all possible keys are tried systematically until correct key is found. This method is used when any other weakness is not useful. The key length used in the encryption process specifies the practical feasibility of brute force attack, with longer keys exponential more difficult to crack as compared to smaller keys[1].One of the measure strength of the encryption system depends on theoretically how much time is taken to mount a successful brute force attack. The resources required for brute force attack grow exponentially with increase in key size, not linearly.

1.11.3. Davies' Attack

This attack is dedicated statistical cryptanalysis method for attacking Data Encryption Standard(DES).This attack was originally created by Donald Davies in 1987.It is a Known Plaintext Attack which is based on non uniform distribution of output of pairs of adjacent S-boxes. It works by collecting various plaintext/cipher text pairs and calculating empirical distribution of its characteristics. Various bits of keys are find out from plaintexts, leaving remaining bits to be find out through brute force attack. There is tradeoff between number of plaintext, keys found and probability of success.

1.11.4. Differential Cryptanalysis

This attack is a chosen plaintext attack in which relationship is find out between the cipher text produced by two related plaintext. It focuses on the statistical analysis of two inputs and two outputs of cryptographic algorithm[4].This scheme can successfully crack DES with an effort on the order of 2^{47} chosen plaintext. In the method, the difference can be specified in several ways but exclusive-OR(XOR) operation is mostly used. The cryptanalyst then encrypts plaintext and its

XORed pairs using all possible sub keys, and it seeks the signs of non- randomness in each pair of intermediate cipher text pairs.

1.11.5. Integral cryptanalysis

This attack is applicable on block cipher based on substitution-permutation networks. Unlike differential cryptanalysis, it uses sets or even multi sets of chosen plaintext of which part is held constant and other part varies with all possibilities It is commonly known as Square attack.

1.11.6. Linear Cryptanalysis

This is a known plaintext attack that require access to large amount of plaintext and cipher text pairs which are encrypted with unknown keys. It focuses on statistical analysis against one round of decryption on large number of cipher text. the attacker decrypts each cipher text using all possible sub keys for one round of encryption and studies the resulting intermediate cipher text to seek the least random result. A sub key which generate the least random intermediate cipher for all cipher texts becomes a candidate key(most likely sub key).

1.11.7. Man-in-the-Middle Attack

This type of attack can be used in those cases in which multiple keys are used for encryption[4].This attack is known plain text attack, the attacker has access to both the plaintext and resulting cipher text. Example is attack versus Double DES. To improve the strength of 56-bit DES, Double DES (two rounds of DES encryption using two different keys, of total key length of 112 bits)was suggested. The attacker wants to recover two keys (key1 and key2) used for encryption. The attacker first apply brute force attack on key1 using all 2^{56} different single keys to encrypt the plaintext and saves each keys and cipher text ant analyst again brute force for key2 by using 2^{56}.The brute force attack is complete when both keys are known to attacker. The attack takes 2^{56} plus at most 2^{56} attack, or maximum 2^{57} total attempts. This is far easier than 2^{112} attempts.

1.12 Cryptanalysis Of Asymmetric Cipher

Asymmetric cryptography is a type which relies on two keys, one private key for decryption and one public key for encryption. Such kind of cipher rely on the "hard" mathematical problem for their security. So the main point of attack is to develop methods to solve such problems. The security of two key cryptography depends on mathematical questions in a way that one key cryptography doesn't, conversely links to wider area of mathematical research in a new way. Asymmetric techniques are designed around of solving various mathematical problems. In case any improved algorithm is found to solve the problem then system is weakened. For example the security of Diffie-Hellman key exchange depends on calculating the discrete logarithm[2].RSA's security depends on difficulty of integer factorization-a breakthrough in factoring would impact security of RSA. Another main feature of asymmetric over symmetric cipher is that cryptanalyst has an opportunity to make use of knowledge obtained from public key.

1.13 Meet In The Middle Attack [7]

A meet-in-the-middle attack is a cryptographic attack, first developed by Diffie and Hellman, that employs a space-time tradeoff to drastically reduce the complexity of cracking a multiple encryption scheme. To illustrate how the attack works, we shall take a look at an example.

Let E_K and D_K denote encryption and decryption functions using the key $K \in \{0,1\}^n$. Similarly, let E'_K and D'_K denote encryption and decryption functions using the key $K \in \{0,1\}^m$. Consider the following simple double-encryption scheme which computes a cipher text message C from a plaintext message P using two keys $K_1 \in \{0,1\}^n$ and $K_2 \in \{0,1\}^m$:

$$C = E'_{K2} (E_{K1} (P))$$

$$P = D_{K1} (D'_{K2} (C))$$

A naive attack on this double-encryption scheme, covering the entire search space of $\{0,1\}^n * \{0,1\}^m$, would require $O(2^{n+m})$ encryptions. However, exhaustive searches to crack E_K and E'_K individually would only take $O(2^n)$ and $O(2^m)$ encryptions, respectively. There is an important derivation from this double-encryption scheme that we can exploit to construct a more sophisticated attack.

$$C = E'_{K2} (E_{K1} (P))$$

$$D'_{K2} (C) = D'_{K2} (E'_{K2} (E_{K1} (P)))$$

$$D'_{K2} (C) = E_{K1} (P)$$

This derivation meets in the middle of the double-encryption scheme and allows us to use exhaustive searches over E_K and E'_K in a more efficient chosen-plaintext attack. Consider one possible approach based on computing the following sets:

$$H = \{(K,E_K(P)) : K \in \{0,1\}^n\}$$

$$S = \{(K_i{:}K_j) : K_i \in \{0,1\}^n \wedge K_j \in \{0,1\}^m \wedge (K_i, D'_{Kj}(C)) \in H\}$$

Here, we precompute the set of all possible encryptions of the plaintext P using EK and store a lookup table H. Afterwards, we compute the set of all possible decryptions of the ciphertext C using D'_K and check for membership in the lookup table. The intersections between the two described sets will contain the correct key pair $(K_1;K_2)$. If there are multiple key pairs in the intersection, then we can test the candidate key pairs using additional plaintext-ciphertext pairs and quickly isolate the correct key pair. This constitutes a much more efficient attack on this double-encryption scheme.

This meet-in-the-middle attack requires $O(2^n + 2^m)$ encryptions to compute the two sets instead of the $O(2^{n+m})$ encryptions required by an exhaustive search. We do incur $O(2^n)$ or $O(2^m)$ space overhead, depending on the approach, in storing the lookup table; however, with modern resources, the space overhead is typically not unreasonable. Meeting in the middle reduces the search space drastically and points out that cracking the double-encryption scheme is computationally similar to cracking the encryption functions that compose it. The math becomes even more alarming in the case where n = m, as this discrepancy becomes $O(2^{2n})$ encryptions for the naive attack and $O(2^{n+1})$ encryptions for the meet-in-the-middle attack, which is only twice what it would take to crack E_K. For this reason, simple multiple-encryption schemes tend to provide considerably fewer bits of effective security than the actual number of key bits used in the encryption scheme.

Chapter 2

SYSTEM REQUIREMENT SPECIFICATION

The following are the system requirements:

2.1 Hardware Requirements

- 512MB RAM or above
- X86 or above processor
- 2MB Secondary memory or above

2.2 Software Requirements

- Operating System: LINUX, Windows
- Language used: C
- Editor: Code blocks IDE

2.3 Functional Requirements

The functional requirements for the implementation are as follows:

2.3.1 Input Specification

- An input file/string type variable should contain some data. That can be used as plain text for encryption

- 3 Secret key used for encryption should of 64-bits each.

2.3.2 Output Specification

- The second party should know secret key that used for encryption.

- After providing secret key as input, it displays the original plain text.

Chapter 3

LITERATURE REVIEW

Table 3.1. Some Reported Technology based on 3DES

Author	Name	Year	Approach	Result
Mandeep Singh Narula, Simarpreet Singh	Implementation of Triple Data Encryption Standard using Verilog	2014	Design synthesis & implementation in Verilog using ISE Foundation. Simulation using ISE simulator.	The proposed implementation of DES and TDES provide high-speed performance with very compact hardware implementation.

Shaunak S.Ganorkar, Shilpi U. Vishwakarma, Sagar D.Pande	An Information Security Scheme for Cloud based Environment using 3DES Encryption Algorithm	2014	Development of a cloud storage system. Use of symmetric key encryption for data, and for authentication	Project achieved the goals of authentication and providing security on cloud storage
Hamdan.O.Alanazi, B.B.Zaidan, A.A.Zaidan, Hamid A.Jalab, M.Shabbir and Y. Al-Nabhani	New Comparative Study Between DES, 3DES and AES within Nine Factors	2011	Comparison was done using 9 standard evaluation criteria	This proved that AES is better than DES and 3DES

Malik Sikander Hayat Khiyal, Aihab Khan, Khansa Shabbir	Performance Evaluation of Encryption Techniques for Confidentiality of Very Large Databases	2011	The input object is encrypted using symmetric encryption scheme and then the performance of these encryption algorithms will be recorded according to different parameters	AES provides the highest security but if the organization is short of resources, then 3DES provides the best solution
Majithia Sachin, Dinesh Kumar	Implementation and Analysis of AES, DES and Triple DES on GSM Network	2010	Brute force attack scheme with varying key length is used. Program is written in MATLAB	AES proves to be better security than DES and 3DES

3.1 Implementation of 3-DES using Verilog

Narula et al.[] proposed an approach for Design synthesis & implementation in Verilog using ISE Foundation and simulation of DES and 3-DES using ISE simulator.

The result was that the proposed implementation of DES and 3DES provided high-speed performance with very compact hardware implementation

The complete design was synthesized and implemented with the use of verilog using ISE Foundation. Simulation was done by ISE simulator.

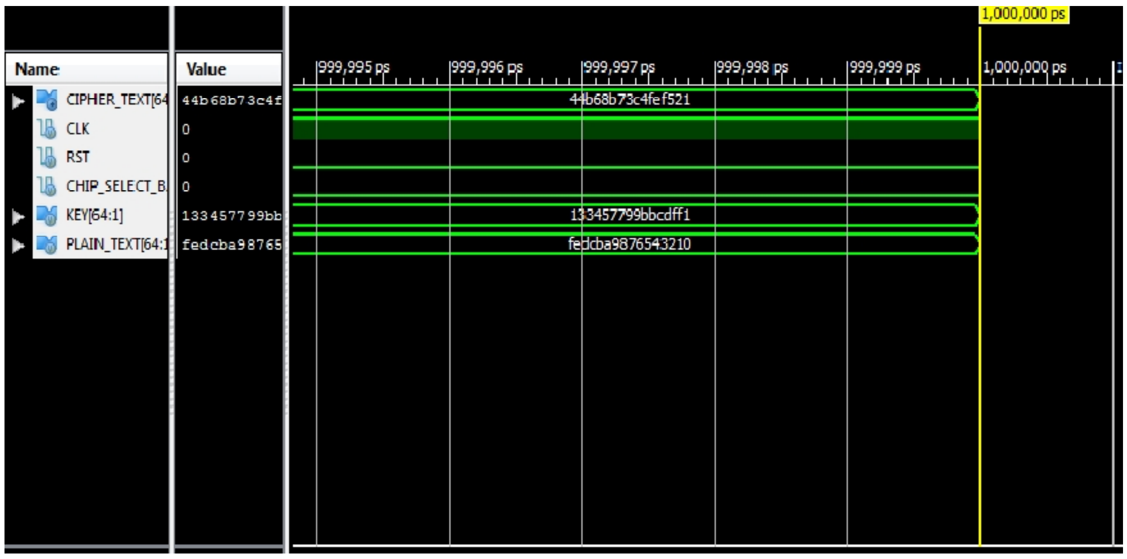


Figure 3.1. Encryption for Single DES Block

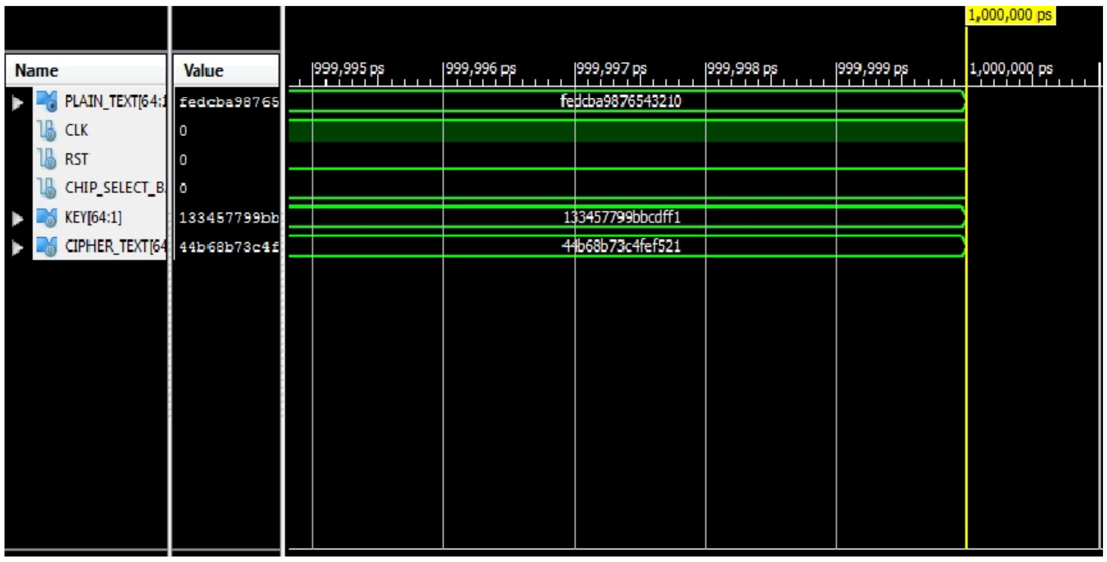
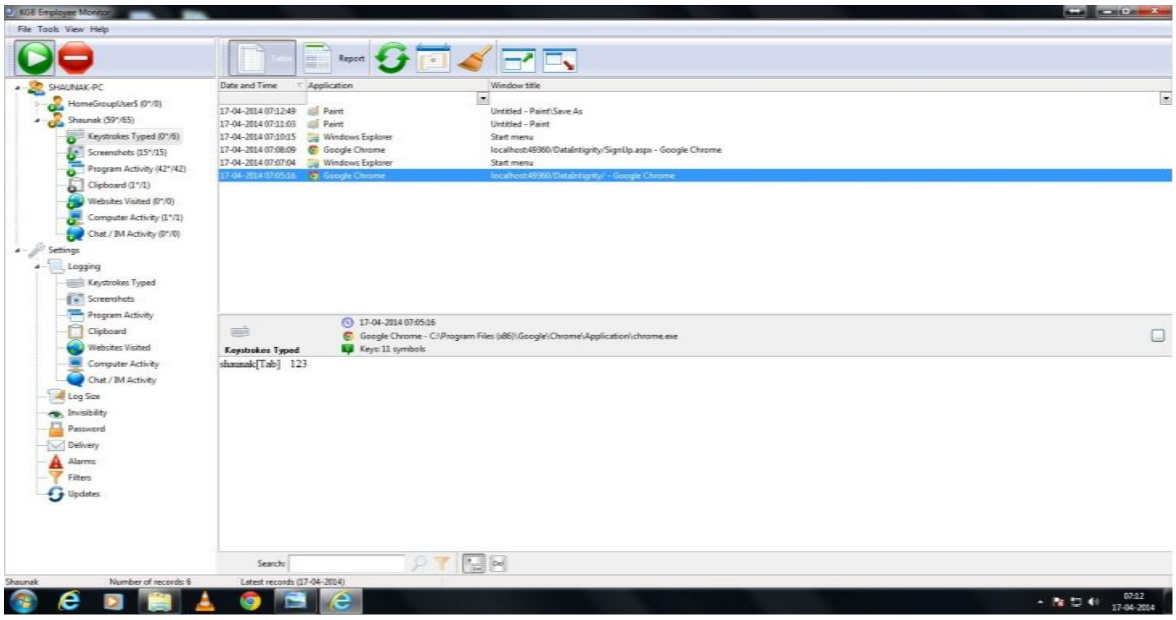


Figure 3.2. Decryption for Single DES block

3.2 An Information Security Scheme for Cloud based Environment using 3DES Encryption Algorithm [8]

Ganorkar et al.[] proposed development of a cloud storage system and use of symmetric key encryption for data, and for authentication.

Given below are the screenshots of the KGB Key logger software which was tested on our project. As shown in the screen shot the first level of password which consists of alphanumeric characters is captured by the software however, it failed to capture the next level of authentication which is graphical password. This result ensured that the KBG Key logger was unable to crack the authentication procedure of our project completely.



Date and Time	Application	Window title
17-04-2014 07:12:49	Paint	Untitled - Paint\Save As
17-04-2014 07:11:03	Paint	Untitled - Paint
17-04-2014 07:10:15	Windows Explorer	Start menu
17-04-2014 07:08:09	Google Chrome	localhost:49360/DataIntigrity/SignUp.aspx - Google Chrome
17-04-2014 07:07:04	Windows Explorer	Start menu
17-04-2014 07:05:16	Google Chrome	localhost:49360/DataIntigrity/ - Google Chrome

Keystrokes Typed

17-04-2014 07:05:16
Google Chrome - C:\Program Files (x86)\Google\Chrome\Application\chrome.exe
Keys: 11 symbols

```
shaunak[Tab]   123
```

Figure 3.3. KGB Key logger

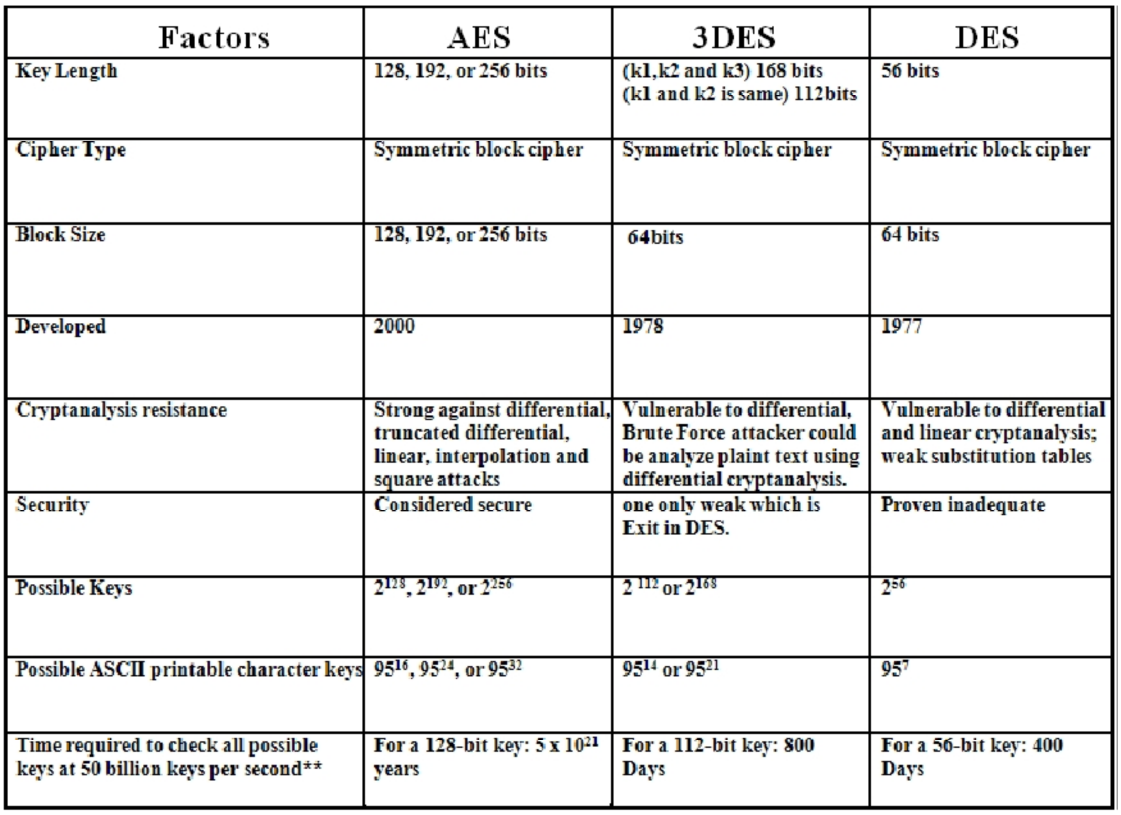
The developed project achieved the goals of authentication and providing security on cloud storage since during our analysis the tools failed to identify the graphical password thus failing to proceed further in our project.

3.3 New Comparative Study Between DES, 3DES and AES within Nine Factors [4]

Alanazi et al.[] proposed a comparitive study between DES, 3DES and AES. This comparison was done using 9 standard evaluation criteria, such as, key length, cipher type, block size, cryptanalysis resistance, security, possible keys, etc.

These eligible's proved that AES is better than DES & 3-DES.

Table 3.2. Comparative Study Between DES, 3DES &AES



Factors	AES	3DES	DES
Key Length	128, 192, or 256 bits	(k1,k2 and k3) 168 bits (k1 and k2 is same) 112bits	56 bits
Cipher Type	Symmetric block cipher	Symmetric block cipher	Symmetric block cipher
Block Size	128, 192, or 256 bits	64bits	64 bits
Developed	2000	1978	1977
Cryptanalysis resistance	Strong against differential, truncated differential, linear, interpolation and square attacks	Vulnerable to differential, Brute Force attacker could be analyze plaint text using differential cryptanalysis.	Vulnerable to differential and linear cryptanalysis; weak substitution tables
Security	Considered secure	one only weak which is Exit in DES.	Proven inadequate
Possible Keys	2^{128}, 2^{192}, or 2^{256}	2^{112} or 2^{168}	2^{56}
Possible ASCII printable character keys	95^{16}, 95^{24}, or 95^{32}	95^{14} or 95^{21}	95^{7}
Time required to check all possible keys at 50 billion keys per second**	For a 128-bit key: 5 x 10^{21} years	For a 112-bit key: 800 Days	For a 56-bit key: 400 Days

Chapter 4

DESIGN AND IMPLEMENTATION

4.1 Algorithm Description

The Triple Data Encryption Standard (3-DES) shall consist of the following Data Encryption Algorithm (DEA) and Triple Data Encryption Algorithm (TDEA, as described in ANSI X9.52). These devices shall be designed in such a way that they may be used in a computer system or network to provide cryptographic protection to binary coded data. The method of implementation will depend on the application and environment. The devices shall be implemented in such a way that they may be tested and validated as accurately performing the transformations specified in the following algorithms.

Data Encryption Algorithm

4.1.1. Introduction [9]

The algorithm is designed to encipher and decipher blocks of data consisting of 64 bits under control of a 64-bit key[1]. Deciphering must be accomplished by using the same key as for enciphering, but with the schedule of addressing the key bits altered so that the deciphering process is the reverse of the enciphering process. A block to be enciphered is subjected to an initial permutation ***IP***, then to a complex key-dependent computation and finally to a permutation which is the inverse of the initial permutation IP^{-1}. The key-dependent computation can be simply defined in terms of a function ***f***, called the cipher function, and a function ***KS***, called the key schedule. A description of the computation is given first, along with details as to how the algorithm is used for encipherment. Next, the use of the algorithm for decipherment is described. Finally, a definition of the cipher function ***f*** is given in terms of primitive functions which are called the selection functions ***Si*** and the permutation function ***P***. ***Si***, ***P*** and ***KS*** of the algorithm are explained later.

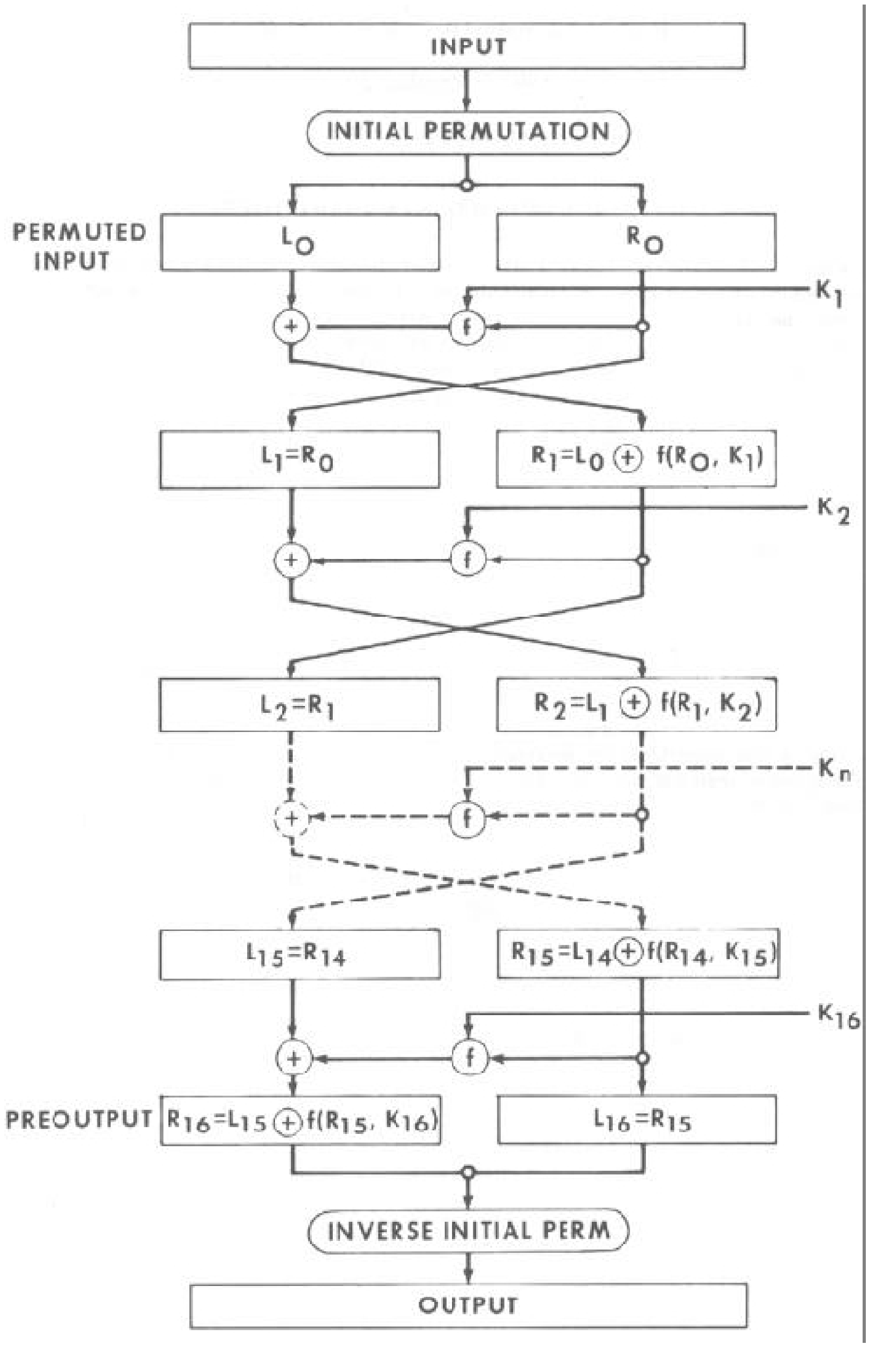


Figure 4.1. Enciphering Computation

The following notation is convenient: Given two blocks ***L*** and ***R*** of bits, ***LR*** denotes the block consisting of the bits of ***L*** followed by the bits of ***R***. Since concatenation is associative, ***B1B2...B8***, for example, denotes the block consisting of the bits of ***B1*** followed by the bits of ***B2***...followed by the bits of ***B8***.

4.1.2. Enciphering

A sketch of the enciphering computation is given in Figure 4.1.

The 64 bits of the input block to be enciphered are first subjected to the following permutation, called the initial permutation ***IP***:

IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

That is the permuted input has bit 58 of the input as its first bit, bit 50 as its second bit, and so on with bit 7 as its last bit. The permuted input block is then the input to a complex key-dependent computation described below. The output of that computation, called the preoutput, is then subjected to the following permutation which is the inverse of the initial permutation:

IP^{-1}

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

That is, the output of the algorithm has bit 40 of the preoutput block as its first bit, bit 8 as its second bit, and so on, until bit 25 of the preoutput block is the last bit of the output.

The computation which uses the permuted input block as its input to produce the preoutput block consists, but for a final interchange of blocks, of 16 iterations of a calculation that is described below in terms of the cipher function ***f*** which operates on two blocks, one of 32 bits and one of 48 bits, and produces a block of 32 bits.

Let the 64 bits of the input block to an iteration consist of a 32 bit block ***L*** followed by a 32 bit block ***R***. Using the notation defined in the introduction, the input block is then ***LR***.

Let ***K*** be a block of 48 bits chosen from the 64-bit key. Then the output ***L'R'*** of an iteration with input ***LR*** is defined by:

$$\begin{aligned} L' &= R \\ R' &= L \oplus f(R,K) \end{aligned} \qquad (1)$$

where Θ denotes bit-by-bit addition modulo 2.

As remarked before, the input of the first iteration of the calculation is the permuted input block.

If ***L'R'*** is the output of the 16th iteration then ***R'L'*** is the preoutput block. At each iteration a different block ***K*** of key bits is chosen from the 64-bit key designated by ***KEY***.

With more notation we can describe the iterations of the computation in more detail. Let K_S be a function which takes an integer *n* in the range from 1 to 16 and a 64-bit block ***KEY*** as input and yields as output a 48-bit block K_n which is a permuted selection of bits from ***KEY***. That is

$$K_n = K_S(n,KEY) \qquad (2)$$

with K_n determined by the bits in 48 distinct bit positions of ***KEY***. K_S is called the key schedule because the block ***K*** used in the n^th iteration of (1) is the block K_n determined by (2).

As before, let the permuted input block be ***LR***. Finally, let ***L()*** and ***R()*** be respectively ***L*** and ***R*** and let L_n and R_n be respectively ***L'*** and ***R'*** of (1) when ***L*** and ***R*** are respectively L_{n-1} and R_{n-1} and ***K*** is K_n; that is, when *n* is in the range from 1 to 16,

$$L_n = R_{n-1}\ R_n = L_{n-1}\ \Theta\ f(R_{n-1},K_n) \qquad (3)$$

The preoutput block is then $R_{16}L_{16}$.

The key schedule K_S of the algorithm is described later in detail. The key schedule produces the 16 K_n which are required for the algorithm.

4.1.3. Deciphering

The permutation $\boldsymbol{IP^{-1}}$ applied to the preoutput block is the inverse of the initial permutation ***IP*** applied to the input. Further, from (1) it follows that:

$$\boldsymbol{R = L' \ L = R' \ \Theta \ f(L',K)} \qquad (4)$$

Consequently, to ***decipher*** it is only necessary to apply the ***very same algorithm to an enciphered message block,*** taking care that at each iteration of the computation ***the same block of key bits*** **K** ***is used*** during decipherment as was used during the encipherment of the block. Using the notation of the previous section, this can be expressed by the equations:

$$\boldsymbol{R_{n-1} = L_n \ L_{n-1} = R_n \ \Theta \ f(L_n,K_n)} \qquad (5)$$

where now $\boldsymbol{R_{16}L_{16}}$ is the permuted input block for the deciphering calculation and $\boldsymbol{L_0R_0}$ is the preoutput block. That is, for the decipherment calculation with $\boldsymbol{R_{16}L_{16}}$ as the permuted input, $\boldsymbol{K_{16}}$ is used in the first iteration, $\boldsymbol{K_{15}}$ in the second, and so on, with $\boldsymbol{K_1}$ used in the 16th iteration.

4.1.4. The Cipher Function f

A sketch of the calculation of ***f(R,K)*** is given in Figure 4.2.

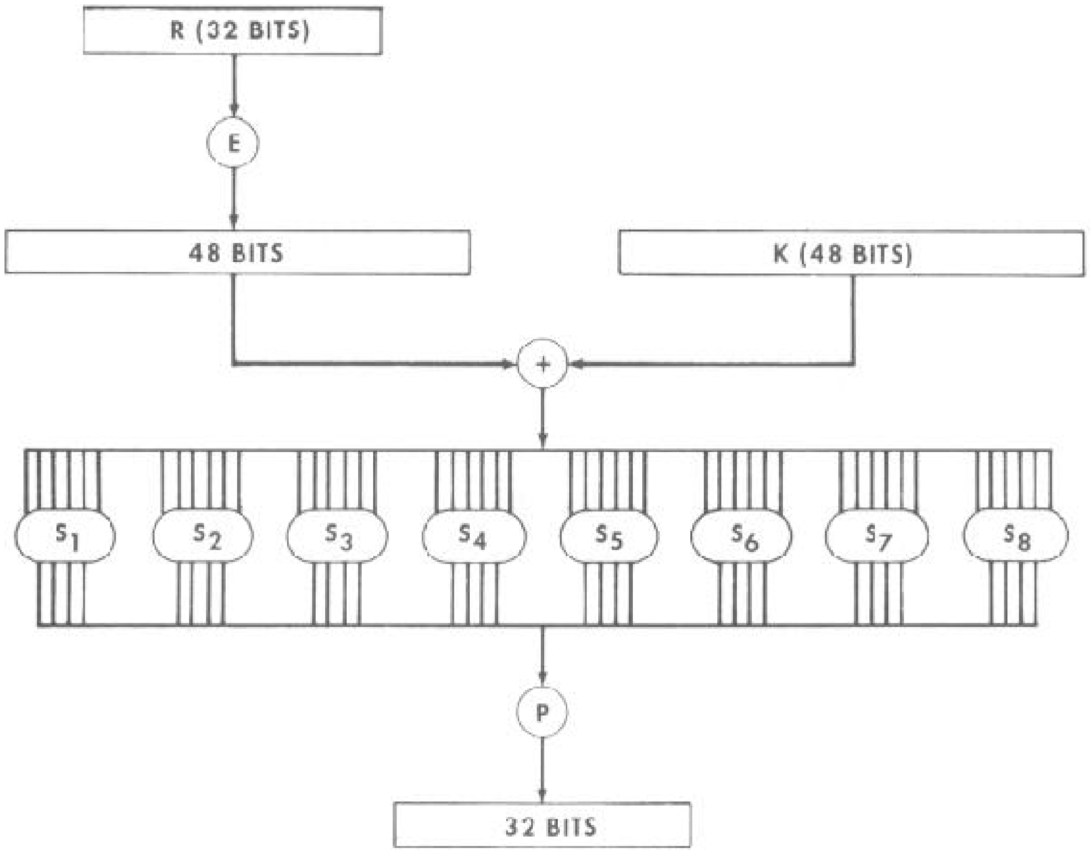


Figure 4.2. Cipher Function f

Let ***E*** denote a function which takes a block of 32 bits as input and yields a block of 48 bits as output. Let ***E*** be such that the 48 bits of its output, written as 8 blocks of 6 bits each, are obtained by selecting the bits in its inputs in order according to the following table:

E BIT-SELECTION TABLE

32	1	2	3	4	5
4	5	6	7	8	9

8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Thus the first three bits of ***E(R)*** are the bits in positions 32, 1 and 2 of ***R*** while the last 2 bits of ***E(R)*** are the bits in positions 32 and 1.

Each of the unique selection functions ***S1,S2,...,S8***, takes a 6-bit block as input and yields a 4-bit block as output and is illustrated by using a table containing the recommended ***S1***:

S1

Column Number

Row	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

If ***S1*** is the function defined in this table and ***B*** is a block of 6 bits, then ***S1(B)*** is determined as follows: The first and last bits of ***B*** represent in base 2 a number in the range 0 to 3. Let that number be *i*. The middle 4 bits of ***B*** represent in base 2 a number in the range 0 to 14. Let that number be *j*. Look up in the table the number in the *i*'th row and *j*'th column. It is a number in the range 0 to 15 and is uniquely represented by a 4 bit block. That block is the output ***S1(B)*** of ***S1*** for the input ***B***. For example, for input 011011 the row is 01, that is row 1, and the column is determined by 1101, that is column 13. In row 1 column 13 appears 5 so that the output is 0101.
Selection functions ***S1,S2,...,S8*** of the algorithm appear later.

The permutation function ***P*** yields a 32-bit output from a 32-bit input by permuting the bits of the input block. Such a function is defined by the following table:

P

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

The output ***P(L)*** for the function ***P*** defined by this table is obtained from the input ***L*** by taking the 16th bit of ***L*** as the first bit of ***P(L)***, the 7th bit as the second bit of ***P(L)***, and so on until the 25th bit of ***L*** is taken as the 32nd bit of ***P(L)***. The permutation function ***P*** of the algorithm is repeated in DEA description.

Now let ***S1,...,S8*** be eight distinct selection functions, let ***P*** be the permutation function and let ***E*** be the function defined above.

To define ***f(R,K)*** we first define ***B1,...,B8*** to be blocks of 6 bits each for which

(6) $$B1B2...B8 = K \ \Theta\ E(R)$$

The block ***f(R,K)*** is then defined to be

(7) $$P(S1(B1)S2(B2)...S8(B8))$$

Thus ***K*** Θ ***E(R)*** is first divided into the 8 blocks as indicated in (6). Then each ***Bi*** is taken as an input to ***Si*** and the 8 blocks ***S1(B1),S2(B2),...,S8(B8)*** of 4 bits each are consolidated into a single block of 32 bits which forms the input to ***P***. The output (7) is then the output of the function ***f*** for the inputs ***R*** and ***K***.

Triple Data Encryption Algorithm [2]

Let $\boldsymbol{E_K(I)}$ and $\boldsymbol{D_K(I)}$ represent the DES encryption and decryption of $\boldsymbol{I}$ using DES key $\boldsymbol{K}$ respectively. Each TDEA encryption/decryption operation (as specified in ANSI X9.52) is a compound operation of DES encryption and decryption operations. The following operations are used:

1. TDEA encryption operation: the transformation of a 64-bit block $\boldsymbol{I}$ into a 64-bit block $\boldsymbol{O}$ that is defined as follows:

$$\boldsymbol{O = E_{K3}(D_{K2}(E_{K1}(I))).}$$

2. TDEA decryption operation: the transformation of a 64-bit block $\boldsymbol{I}$ into a 64-bit block $\boldsymbol{O}$ that is defined as follows:

$$\boldsymbol{O = D_{K1}(E_{K2}(D_{K3}(I)))}$$

The standard specifies the following keying options for bundle ***(K1, K2, K3)***

1. Keying Option 1: ***K1, K2*** and ***K3*** are independent keys;

2. Keying Option 2: ***K1*** and ***K2*** are independent keys and ***K3 = K1***;

3. Keying Option 3: ***K1 = K2 = K3***.

A TDEA mode of operation is backward compatible with its single DES counterpart if, with compatible keying options for TDEA operation,

1. an encrypted plaintext computed using a single DES mode of operation can be decrypted correctly by a corresponding TDEA mode of operation; and

2. an encrypted plaintext computed using a TDEA mode of operation can be decrypted correctly by a corresponding single DES mode of operation.

When using Keying Option 3 ***(K1 = K2 = K3)***, TECB, TCBC, TCFB and TOFB modes are backward compatible with single DES modes of operation ECB, CBC, CFB, OFB respectively.

The diagram drawn below illustrates TDEA encryption and TDEA decryption.

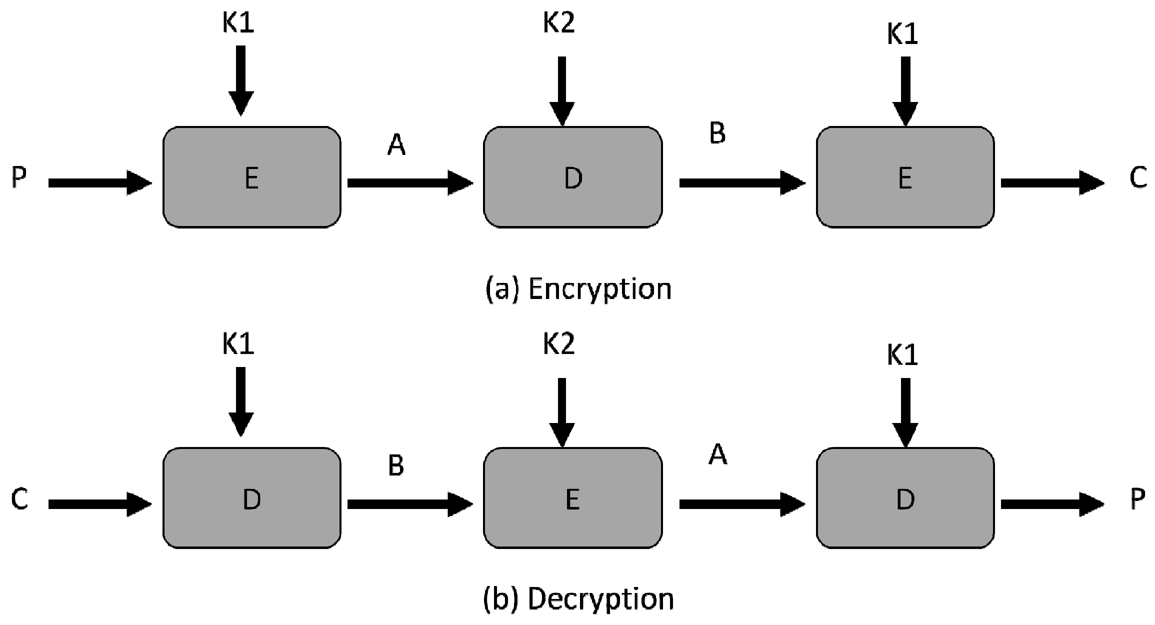


Figure 4.3. Triple DES Block Diagram

Data Encryption Algorithm (Conti...)

The choice of the primitive functions ***KS, S1,...,S8*** and ***P*** is critical to the strength of an encipherment resulting from the algorithm. Specified below is the recommended set of functions, describing ***S1,...,S8*** and ***P*** in the same way they are described in the algorithm. For the interpretation of the tables describing these functions, see the discussion in the body of the algorithm.

The primitive functions ***S1,...,S8*** are:

S1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	1

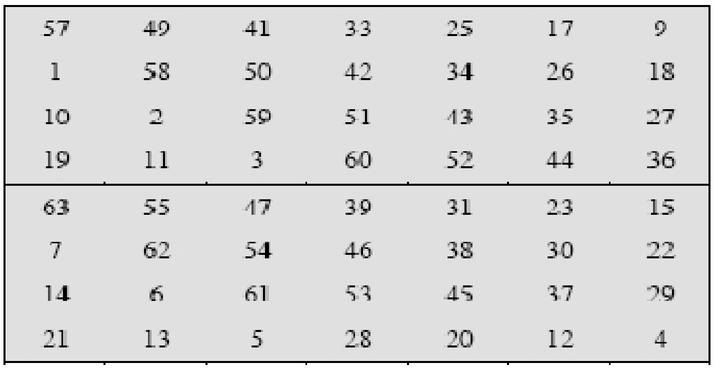
The primitive function ***P*** is:

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Recall that $\boldsymbol{K_n}$, for 1<=*n*<=16, is the block of 48 bits in (2) of the algorithm. Hence, to describe $\boldsymbol{K_S}$, it is sufficient to describe the calculation of $\boldsymbol{K_n}$ from ***KEY*** for *n* = 1, 2,..., 16. To complete the definition of $\boldsymbol{K_S}$ it is therefore sufficient to describe the two permuted choices, as well as the schedule of left shifts. One bit in each 8-bit byte of the ***KEY*** may be utilized for error detection in key generation, distribution and storage. Bits 8, 16,..., 64 are for use in assuring that each byte is of odd parity.

Permuted choice 1 is determined by the following table:

***PC*-1**



57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

The table has been divided into two parts, with the first part determining how the bits of ***C( )*** are chosen, and the second part determining how the bits of ***D( )*** are chosen. The bits of ***KEY*** are numbered 1 through 63. The bits of ***C( )*** are respectively bits 57, 49, 41,..., 44 and 36 of ***KEY***, with the bits of ***D( )*** being bits 63, 55, 47,..., 12 and 4 of ***KEY***.

With ***C( )*** and ***D***( ) defined, we now define how the blocks $\boldsymbol{C_n}$ and $\boldsymbol{D_n}$ are obtained from the blocks $\boldsymbol{C_{n1}}$ and $\boldsymbol{D_{n-1}}$, respectively, for *n* = 1, 2,..., 16. That is accomplished by adhering to the following schedule of left shifts of the individual blocks:

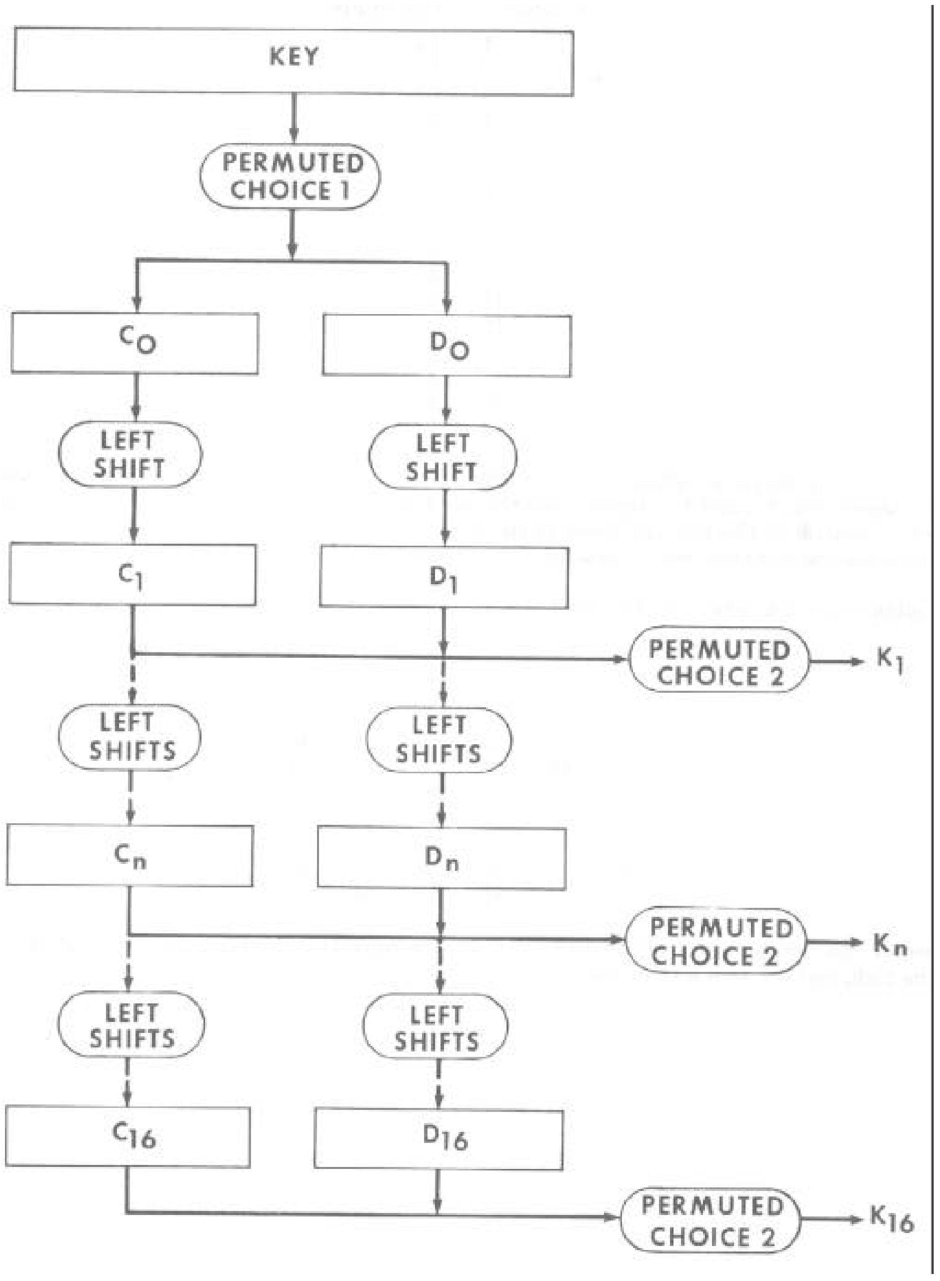


Figure 4.3. Fiestal Structure

Iteration Number	***Number of Left Shifts***
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

For example, $\boldsymbol{C_3}$ and $\boldsymbol{D_3}$ are obtained from $\boldsymbol{C_2}$ and $\boldsymbol{D_2}$, respectively, by two left shifts, and $\boldsymbol{C_{16}}$ and $\boldsymbol{D_{16}}$ are obtained from $\boldsymbol{C_{15}}$ and $\boldsymbol{D_{15}}$, respectively, by one left shift. In all cases, by a single left shift is meant a rotation of the bits one place to the left, so that after one left shift the bits in the 28 positions are the bits that were previously in positions 2, 3,..., 28, 1.

Permuted choice 2 is determined by the following table:

PC-2

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Therefore, the first bit of K_n is the 14th bit of C_nD_n, the second bit the 17th, and so on with the 47th bit the 29th, and the 48th bit the 32nd.

4.2 Source Code

```
#define uchar unsigned char
#define uint unsigned int
#define ENCRYPT 1
#define DECRYPT 0

// Obtain bit "b" from the left and shift it "c" places from the right
#define BITNUM(a,b,c) (((a[(b)/8] >> (7 - (b%8))) & 0x01) << (c))
#define BITNUMINTR(a,b,c) ((((a) >> (31 - (b))) & 0x00000001) << (c))
#define BITNUMINTL(a,b,c) ((((a) << (b)) & 0x80000000) >> (c))
// This macro converts a 6 bit block with the S-Box row defined as the first and last
// bits to a 6 bit block with the row defined by the first two bits.
#define SBOXBIT(a) (((a) & 0x20) | (((a) & 0x1f) >> 1) | (((a) & 0x01) << 4))

uchar sbox1[64] = {
  14,  4,  13,  1,   2, 15,  11,  8,   3, 10,   6, 12,   5,  9,   0,  7,
   0, 15,   7,  4,  14,  2,  13,  1,  10,  6,  12, 11,   9,  5,   3,  8,
   4,  1,  14,  8,  13,  6,   2, 11,  15, 12,   9,  7,   3, 10,   5,  0,
  15, 12,   8,  2,   4,  9,   1,  7,   5, 11,   3, 14,  10,  0,   6, 13
};

uchar sbox2[64] = {
  15,  1,   8, 14,   6, 11,   3,  4,   9,  7,   2, 13,  12,  0,   5, 10,
   3, 13,   4,  7,  15,  2,   8, 14,  12,  0,   1, 10,   6,  9,  11,  5,
   0, 14,   7, 11,  10,  4,  13,  1,   5,  8,  12,  6,   9,  3,   2, 15,
  13,  8,  10,  1,   3, 15,   4,  2,  11,  6,   7, 12,   0,  5,  14,  9
};
```

```
uchar sbox3[64] = {
   10,  0,   9, 14,   6,  3,  15,  5,   1, 13,  12,  7,  11,  4,   2,  8,
   13,  7,   0,  9,   3,  4,   6, 10,   2,  8,   5, 14,  12, 11,  15,  1,
   13,  6,   4,  9,   8, 15,   3,  0,  11,  1,   2, 12,   5, 10,  14,  7,
    1, 10,  13,  0,   6,  9,   8,  7,   4, 15,  14,  3,  11,  5,   2, 12
};

uchar sbox4[64] = {
    7, 13,  14,  3,   0,  6,   9, 10,   1,  2,   8,  5,  11, 12,   4, 15,
   13,  8,  11,  5,   6, 15,   0,  3,   4,  7,   2, 12,   1, 10,  14,  9,
   10,  6,   9,  0,  12, 11,   7, 13,  15,  1,   3, 14,   5,  2,   8,  4,
    3, 15,   0,  6,  10,  1,  13,  8,   9,  4,   5, 11,  12,  7,   2, 14
};

uchar sbox5[64] = {
    2, 12,   4,  1,   7, 10,  11,  6,   8,  5,   3, 15,  13,  0,  14,  9,
   14, 11,   2, 12,   4,  7,  13,  1,   5,  0,  15, 10,   3,  9,   8,  6,
    4,  2,   1, 11,  10, 13,   7,  8,  15,  9,  12,  5,   6,  3,   0, 14,
   11,  8,  12,  7,   1, 14,   2, 13,   6, 15,   0,  9,  10,  4,   5,  3
};

uchar sbox6[64] = {
   12,  1,  10, 15,   9,  2,   6,  8,   0, 13,   3,  4,  14,  7,   5, 11,
   10, 15,   4,  2,   7, 12,   9,  5,   6,  1,  13, 14,   0, 11,   3,  8,
    9, 14,  15,  5,   2,  8,  12,  3,   7,  0,   4, 10,   1, 13,  11,  6,
    4,  3,   2, 12,   9,  5,  15, 10,  11, 14,   1,  7,   6,  0,   8, 13
};
```

```
uchar sbox7[64] = {
    4, 11,   2, 14,  15,  0,   8, 13,   3, 12,   9,  7,   5, 10,   6,  1,
   13,  0,  11,  7,   4,  9,   1, 10,  14,  3,   5, 12,   2, 15,   8,  6,
    1,  4,  11, 13,  12,  3,   7, 14,  10, 15,   6,  8,   0,  5,   9,  2,
    6, 11,  13,  8,   1,  4,  10,  7,   9,  5,   0, 15,  14,  2,   3, 12
};

uchar sbox8[64] = {
   13,  2,   8,  4,   6, 15,  11,  1,  10,  9,   3, 14,   5,  0,  12,  7,
    1, 15,  13,  8,  10,  3,   7,  4,  12,  5,   6, 11,   0, 14,   9,  2,
    7, 11,   4,  1,   9, 12,  14,  2,   0,  6,  10, 13,  15,  3,   5,  8,
    2,  1,  14,  7,   4, 10,   8, 13,  15, 12,   9,  0,   3,  5,   6, 11
};


void key_schedule(uchar key[], uchar schedule[][6], uint mode)
{
   uint i,j,to_gen,C,D,
        key_rnd_shift[16]={1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1},
        key_perm_c[28]={56,48,40,32,24,16,8,0,57,49,41,33,25,17,
                        9,1,58,50,42,34,26,18,10,2,59,51,43,35},
        key_perm_d[28]={62,54,46,38,30,22,14,6,61,53,45,37,29,21,
                        13,5,60,52,44,36,28,20,12,4,27,19,11,3},
        key_compression[48]={13,16,10,23,0,4,2,27,14,5,20,9,
                             22,18,11,3,25,7,15,6,26,19,12,1,
                             40,51,30,36,46,54,29,39,50,44,32,47,
                             43,48,38,55,33,52,45,41,49,35,28,31};
```

```
 // Permutated Choice #1 (copy the key in, ignoring parity bits).
  for (i = 0, j = 31, C = 0; i < 28; ++i, --j)
     C |= BITNUM(key,key_perm_c[i],j);
  for (i = 0, j = 31, D = 0; i < 28; ++i, --j)
     D |= BITNUM(key,key_perm_d[i],j);


  // Generate the 16 subkeys.
  for (i = 0; i < 16; ++i) {
     C = ((C << key_rnd_shift[i]) | (C >> (28-key_rnd_shift[i]))) & 0xfffffff0;
     D = ((D << key_rnd_shift[i]) | (D >> (28-key_rnd_shift[i]))) & 0xfffffff0;

     // Decryption subkeys are reverse order of encryption subkeys so
     // generate them in reverse if the key schedule is for decryption useage.
     if (mode == DECRYPT)
        to_gen = 15 - i;
     else
        to_gen = i;
     // Initialize the array
     for (j = 0; j < 6; ++j)
        schedule[to_gen][j] = 0;
     for (j = 0; j < 24; ++j)
        schedule[to_gen][j/8] |= BITNUMINTR(C,key_compression[j],7 - (j%8));
     for ( ; j < 48; ++j)
        schedule[to_gen][j/8] |= BITNUMINTR(D,key_compression[j] - 28,7 - (j%8));
  }
}
```

```
// Initial (Inv)Permutation step
void IP(uint state[], uchar in[])
{
  state[0] = BITNUM(in,57,31) | BITNUM(in,49,30) | BITNUM(in,41,29) |
BITNUM(in,33,28) |
          BITNUM(in,25,27) | BITNUM(in,17,26) | BITNUM(in,9,25) |
BITNUM(in,1,24) |
          BITNUM(in,59,23) | BITNUM(in,51,22) | BITNUM(in,43,21) |
BITNUM(in,35,20) |
          BITNUM(in,27,19) | BITNUM(in,19,18) | BITNUM(in,11,17) |
BITNUM(in,3,16) |
          BITNUM(in,61,15) | BITNUM(in,53,14) | BITNUM(in,45,13) |
BITNUM(in,37,12) |
          BITNUM(in,29,11) | BITNUM(in,21,10) | BITNUM(in,13,9) | BITNUM(in,5,8)|
          BITNUM(in,63,7) | BITNUM(in,55,6) | BITNUM(in,47,5) | BITNUM(in,39,4) |
          BITNUM(in,31,3) | BITNUM(in,23,2) | BITNUM(in,15,1) | BITNUM(in,7,0);

  state[1] = BITNUM(in,56,31) | BITNUM(in,48,30) | BITNUM(in,40,29) |
BITNUM(in,32,28) |
          BITNUM(in,24,27) | BITNUM(in,16,26) | BITNUM(in,8,25) |
BITNUM(in,0,24) |
          BITNUM(in,58,23) | BITNUM(in,50,22) | BITNUM(in,42,21) |
BITNUM(in,34,20) |
          BITNUM(in,26,19) | BITNUM(in,18,18) | BITNUM(in,10,17) |
BITNUM(in,2,16) |
          BITNUM(in,60,15) | BITNUM(in,52,14) | BITNUM(in,44,13) |
BITNUM(in,36,12) |
          BITNUM(in,28,11) | BITNUM(in,20,10) | BITNUM(in,12,9) | BITNUM(in,4,8)|
          BITNUM(in,62,7) | BITNUM(in,54,6) | BITNUM(in,46,5) | BITNUM(in,38,4) |
          BITNUM(in,30,3) | BITNUM(in,22,2) | BITNUM(in,14,1) | BITNUM(in,6,0);
```

```
}
void InvIP(uint state[], uchar in[])
{
  in[0] = BITNUMINTR(state[1],7,7) | BITNUMINTR(state[0],7,6) |
BITNUMINTR(state[1],15,5) |
        BITNUMINTR(state[0],15,4) | BITNUMINTR(state[1],23,3) |
BITNUMINTR(state[0],23,2) |
        BITNUMINTR(state[1],31,1) | BITNUMINTR(state[0],31,0);

  in[1] = BITNUMINTR(state[1],6,7) | BITNUMINTR(state[0],6,6) |
BITNUMINTR(state[1],14,5) |
        BITNUMINTR(state[0],14,4) | BITNUMINTR(state[1],22,3) |
BITNUMINTR(state[0],22,2) |
        BITNUMINTR(state[1],30,1) | BITNUMINTR(state[0],30,0);

  in[2] = BITNUMINTR(state[1],5,7) | BITNUMINTR(state[0],5,6) |
BITNUMINTR(state[1],13,5) |
        BITNUMINTR(state[0],13,4) | BITNUMINTR(state[1],21,3) |
BITNUMINTR(state[0],21,2) |
        BITNUMINTR(state[1],29,1) | BITNUMINTR(state[0],29,0);

  in[3] = BITNUMINTR(state[1],4,7) | BITNUMINTR(state[0],4,6) |
BITNUMINTR(state[1],12,5) |
        BITNUMINTR(state[0],12,4) | BITNUMINTR(state[1],20,3) |
BITNUMINTR(state[0],20,2) |
        BITNUMINTR(state[1],28,1) | BITNUMINTR(state[0],28,0);

  in[4] = BITNUMINTR(state[1],3,7) | BITNUMINTR(state[0],3,6) |
BITNUMINTR(state[1],11,5) |
        BITNUMINTR(state[0],11,4) | BITNUMINTR(state[1],19,3) |
BITNUMINTR(state[0],19,2) |
        BITNUMINTR(state[1],27,1) | BITNUMINTR(state[0],27,0);

  in[5] = BITNUMINTR(state[1],2,7) | BITNUMINTR(state[0],2,6) |
BITNUMINTR(state[1],10,5) |
        BITNUMINTR(state[0],10,4) | BITNUMINTR(state[1],18,3) |
BITNUMINTR(state[0],18,2) |
        BITNUMINTR(state[1],26,1) | BITNUMINTR(state[0],26,0);

  in[6] = BITNUMINTR(state[1],1,7) | BITNUMINTR(state[0],1,6) |
BITNUMINTR(state[1],9,5) |
        BITNUMINTR(state[0],9,4) | BITNUMINTR(state[1],17,3) |
BITNUMINTR(state[0],17,2) |
        BITNUMINTR(state[1],25,1) | BITNUMINTR(state[0],25,0);

  in[7] = BITNUMINTR(state[1],0,7) | BITNUMINTR(state[0],0,6) |
BITNUMINTR(state[1],8,5) |
        BITNUMINTR(state[0],8,4) | BITNUMINTR(state[1],16,3) |
BITNUMINTR(state[0],16,2) |
        BITNUMINTR(state[1],24,1) | BITNUMINTR(state[0],24,0);
}

uint f(uint state, uchar key[])
{
  uchar lrgstate[6],i;
  uint t1,t2;

  // Expansion Permutation
  t1 = BITNUMINTL(state,31,0) | ((state & 0xf0000000) >> 1) |
BITNUMINTL(state,4,5) |
      BITNUMINTL(state,3,6) | ((state & 0x0f000000) >> 3) | BITNUMINTL(state,8,11)|
        BITNUMINTL(state,7,12) | ((state & 0x00f00000) >> 5) |
BITNUMINTL(state,12,17) |
        BITNUMINTL(state,11,18) | ((state & 0x000f0000) >> 7) |
BITNUMINTL(state,16,23);
   t2 = BITNUMINTL(state,15,0) | ((state & 0x0000f000) << 15) |
BITNUMINTL(state,20,5) |
        BITNUMINTL(state,19,6) | ((state & 0x00000f00) << 13) |
BITNUMINTL(state,24,11) |
        BITNUMINTL(state,23,12) | ((state & 0x000000f0) << 11) |
BITNUMINTL(state,28,17) |
        BITNUMINTL(state,27,18) | ((state & 0x0000000f) << 9) |
BITNUMINTL(state,0,23);

   lrgstate[0] = (t1 >> 24) & 0x000000ff;
   lrgstate[1] = (t1 >> 16) & 0x000000ff;
   lrgstate[2] = (t1 >> 8) & 0x000000ff;
   lrgstate[3] = (t2 >> 24) & 0x000000ff;
   lrgstate[4] = (t2 >> 16) & 0x000000ff;
   lrgstate[5] = (t2 >> 8) & 0x000000ff;

   // Key XOR
   lrgstate[0] ^= key[0];
   lrgstate[1] ^= key[1];
   lrgstate[2] ^= key[2];
   lrgstate[3] ^= key[3];
   lrgstate[4] ^= key[4];
   lrgstate[5] ^= key[5];

   // S-Box Permutation
   state = (sbox1[SBOXBIT(lrgstate[0] >> 2)] << 28) |
           (sbox2[SBOXBIT(((lrgstate[0] & 0x03) << 4) | (lrgstate[1] >> 4))] << 24) |
           (sbox3[SBOXBIT(((lrgstate[1] & 0x0f) << 2) | (lrgstate[2] >> 6))] << 20) |
           (sbox4[SBOXBIT(lrgstate[2] & 0x3f)] << 16) |
           (sbox5[SBOXBIT(lrgstate[3] >> 2)] << 12) |
           (sbox6[SBOXBIT(((lrgstate[3] & 0x03) << 4) | (lrgstate[4] >> 4))] << 8) |
           (sbox7[SBOXBIT(((lrgstate[4] & 0x0f) << 2) | (lrgstate[5] >> 6))] << 4) |
            sbox8[SBOXBIT(lrgstate[5] & 0x3f)];

   // P-Box Permutation
   state = BITNUMINTL(state,15,0) | BITNUMINTL(state,6,1) |
BITNUMINTL(state,19,2) |
           BITNUMINTL(state,20,3) | BITNUMINTL(state,28,4) |
BITNUMINTL(state,11,5) |
           BITNUMINTL(state,27,6) | BITNUMINTL(state,16,7) |
BITNUMINTL(state,0,8) |
           BITNUMINTL(state,14,9) | BITNUMINTL(state,22,10) |
BITNUMINTL(state,25,11) |
           BITNUMINTL(state,4,12) | BITNUMINTL(state,17,13) |
BITNUMINTL(state,30,14) |
           BITNUMINTL(state,9,15) | BITNUMINTL(state,1,16) |
BITNUMINTL(state,7,17) |
           BITNUMINTL(state,23,18) | BITNUMINTL(state,13,19) |
BITNUMINTL(state,31,20) |
           BITNUMINTL(state,26,21) | BITNUMINTL(state,2,22) |
BITNUMINTL(state,8,23) |
           BITNUMINTL(state,18,24) | BITNUMINTL(state,12,25) |
BITNUMINTL(state,29,26) |
           BITNUMINTL(state,5,27) | BITNUMINTL(state,21,28) |
BITNUMINTL(state,10,29) |
           BITNUMINTL(state,3,30) | BITNUMINTL(state,24,31);

   // Return the final state value
```

```
  return(state);
}
void des_crypt(uchar in[], uchar out[], uchar key[][6])
{
  uint state[2],idx,t;
  IP(state,in);

  // Loop 16 times, perform the final loop manually as it doesn't switch sides
  for (idx=0; idx < 15; ++idx) {
    t = state[1];
    state[1] = f(state[1],key[idx]) ^ state[0];
    state[0] = t;
  }
  state[0] = f(state[1],key[15]) ^ state[0];

  // Inverse IP
  InvIP(state,out);
}

/*************************************
         3DES functions
*************************************/

void three_des_key_schedule(uchar key[], uchar schedule[][16][6], uint mode)
{

  if (mode == ENCRYPT) {
    key_schedule(&key[0],schedule[0],mode);
    key_schedule(&key[8],schedule[1],!mode);
    key_schedule(&key[16],schedule[2],mode);
  }
  else {
    key_schedule(&key[16],schedule[0],mode);
    key_schedule(&key[8],schedule[1],!mode);
    key_schedule(&key[0],schedule[2],mode);
  }
}


void three_des_crypt(uchar in[], uchar out[], uchar key[][16][6])
{
  des_crypt(in,out,key[0]);
  des_crypt(out,out,key[1]);
  des_crypt(out,out,key[2]);
}
```

4.3 Performance Parameters

The performance of 3DES algorithm can be measured by considering following parameters:

4.3.1. Time Taken

The time taken for encryption as well as decryption of a given plain text is calculated by as follows using system clock time: The system clock is recorded twice i.e. before and after the execution of the encryption module and their difference yields the time taken for encryption. The same procedure is followed to calculate decryption time, just that decryption module is invoked instead.

4.3.2. Throughput

In computer technology, throughput is the amount of work that a computer can do in a given time period. Throughput is one of the key factors to measure performance of an algorithm. In case of DES, throughput depends on size of block as well as time taken for encryption/decryption given by:

$$T = \frac{block\ size}{t}$$

where,

T- throughput

t- time taken to encrypt/decrypt

4.4 Performance Analysis

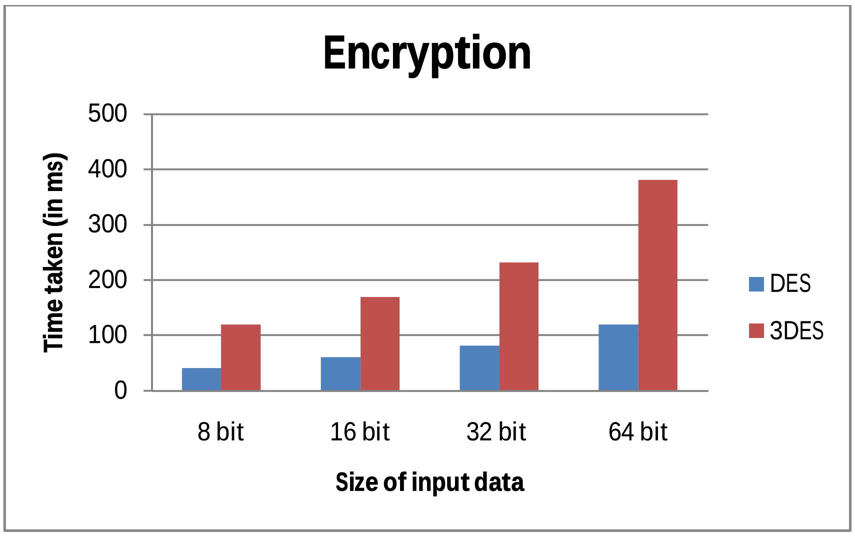


Figure 4.4. Analysis of Time Taken for Encryption

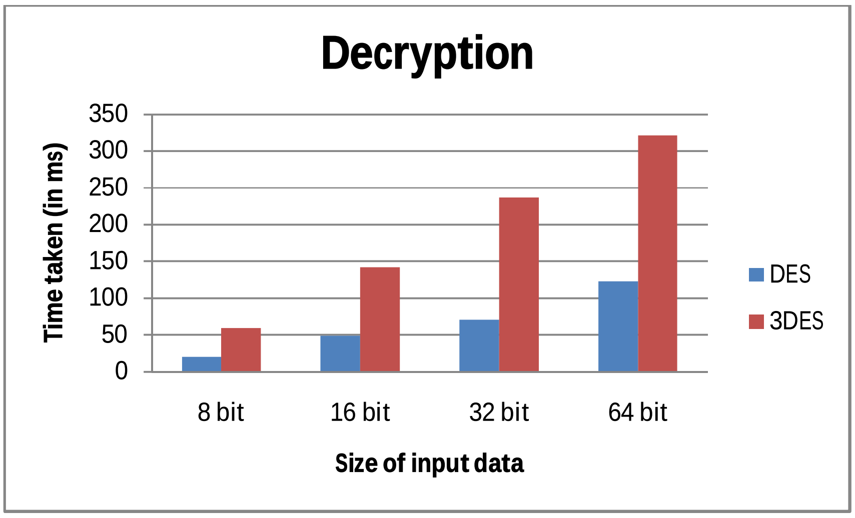


Figure 4.6. Analysis of Time Taken for Decryption

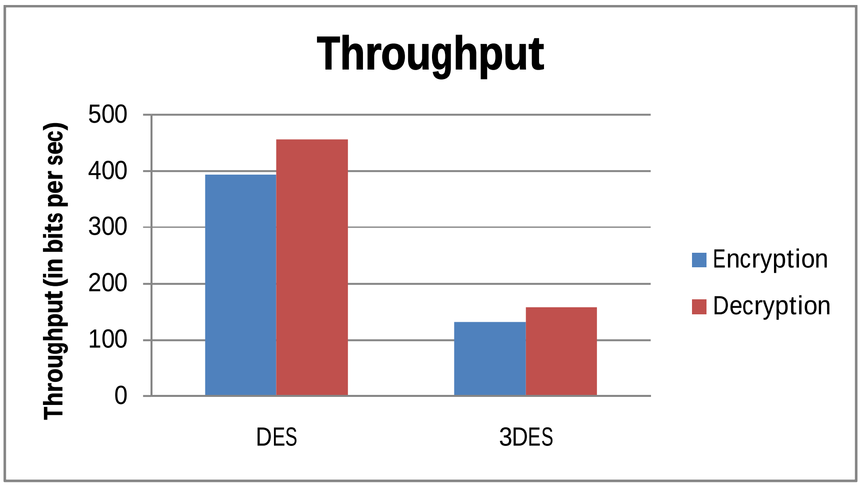
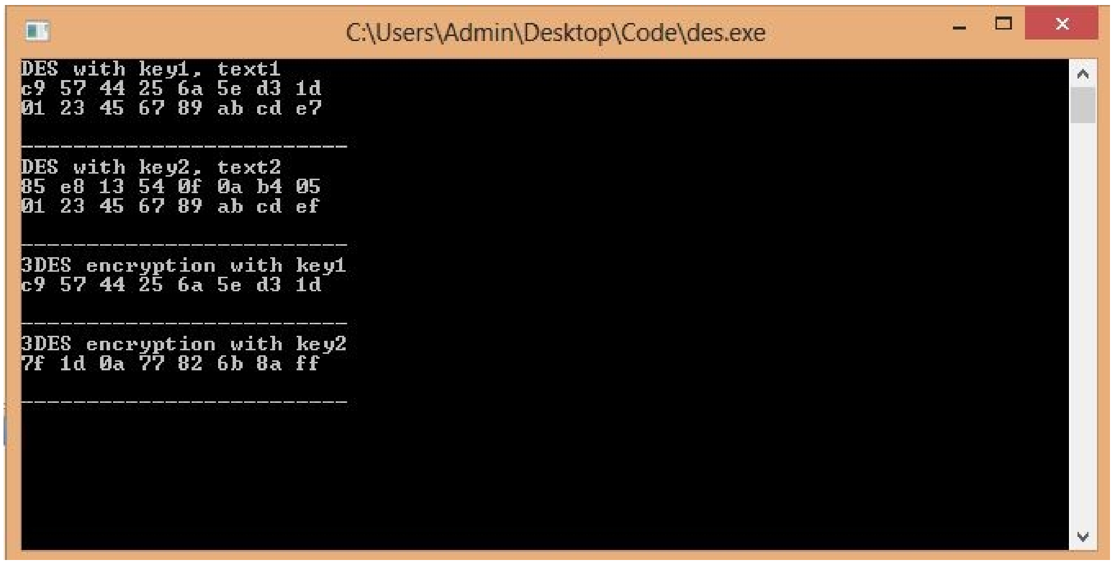


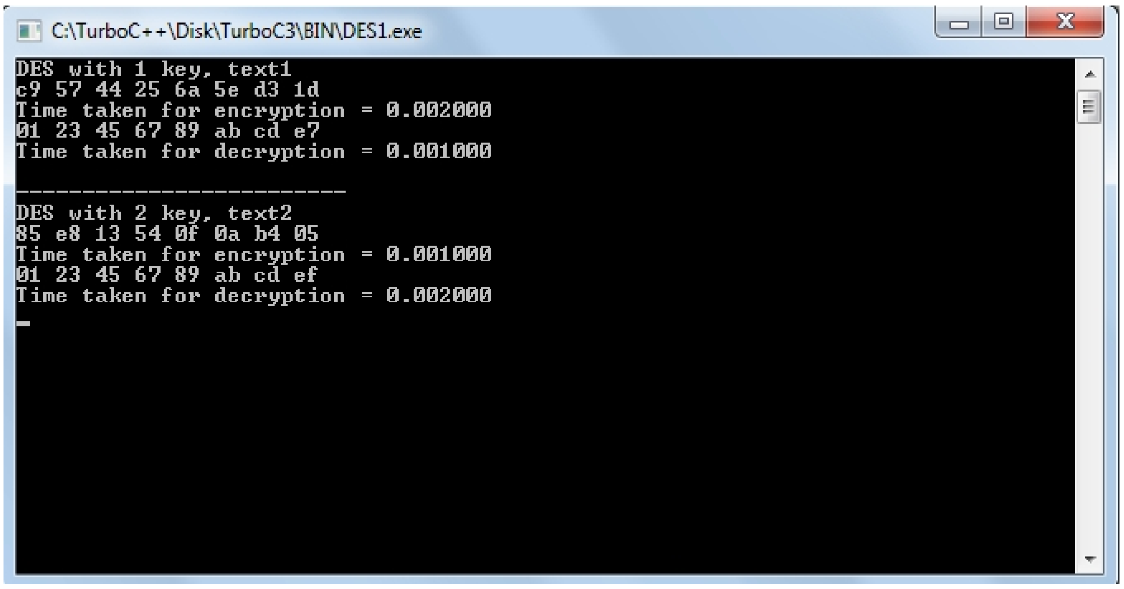
Figure 4.7. Analysis of Throughput

4.5 Output



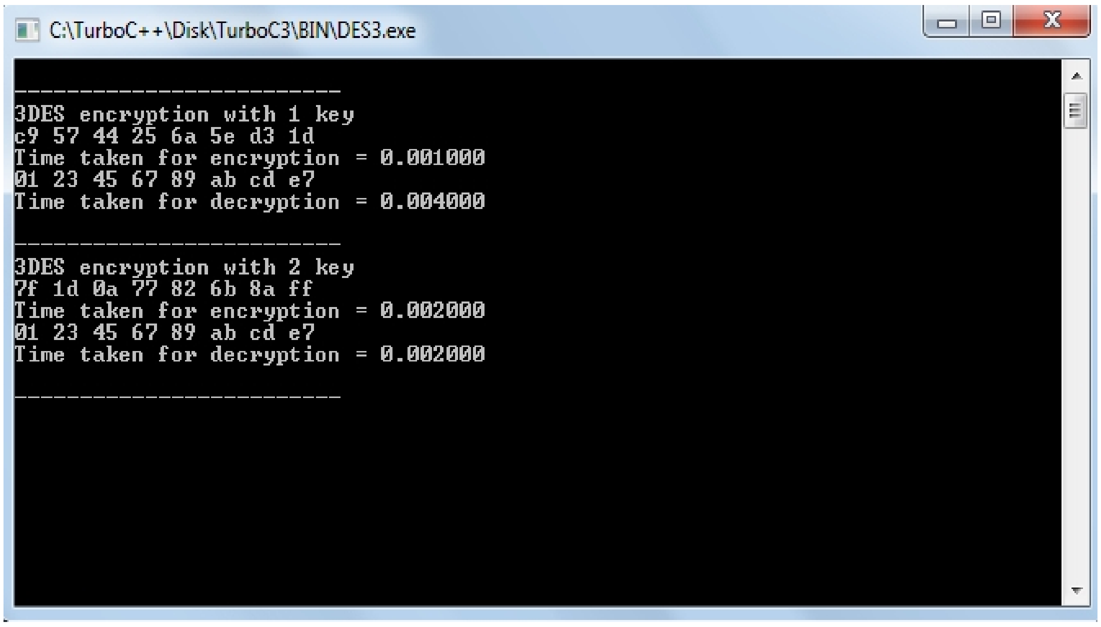
```
DES with key1, text1
c9 57 44 25 6a 5e d3 1d
01 23 45 67 89 ab cd e7

-------------------------
DES with key2, text2
85 e8 13 54 0f 0a b4 05
01 23 45 67 89 ab cd ef

-------------------------
3DES encryption with key1
c9 57 44 25 6a 5e d3 1d

-------------------------
3DES encryption with key2
7f 1d 0a 77 82 6b 8a ff

-------------------------
```

Figure 4.8. Output Screen 1



```
DES with 1 key, text1
c9 57 44 25 6a 5e d3 1d
Time taken for encryption = 0.002000
01 23 45 67 89 ab cd e7
Time taken for decryption = 0.001000

-------------------------
DES with 2 key, text2
85 e8 13 54 0f 0a b4 05
Time taken for encryption = 0.001000
01 23 45 67 89 ab cd ef
Time taken for decryption = 0.002000
```

Figure 4.9. Output Screen 2



```
-------------------------
3DES encryption with 1 key
c9 57 44 25 6a 5e d3 1d
Time taken for encryption = 0.001000
01 23 45 67 89 ab cd e7
Time taken for decryption = 0.004000

-------------------------
3DES encryption with 2 key
7f 1d 0a 77 82 6b 8a ff
Time taken for encryption = 0.002000
01 23 45 67 89 ab cd e7
Time taken for decryption = 0.002000

-------------------------
```

Figure 4.10. Output Screen 3

Chapter 5

Conclusion and Future Work

The developed project achieved the goals of authentication and providing security using the C language. Triple DES was successfully implemented on various data messages with various keys. Furthermore, we assimilated information on the performance of DES and 3DES and conducted a detailed comparison between the two encryption standards. This concludes us in saying that the goals that were set during the development of the project have been achieved as desired.

For the foreseeable future Triple DES is an excellent and reliable choice for the security needs of highly sensitive information. The AES will be at least as strong as Triple DES and probably much faster. It's the industry mandate from Visa and MasterCard that's requiring ATM deployers to upgrade and/or replace their legacy terminals. In a nutshell, it's all about three waves of encryption, and it's designed to make ATM transactions more secure.

Further improvement to 3DES implementation can be done by harnessing MMX and SIMD on modern CPUs as well as pushing down this computation to normal graphic cards using NVIDIA CUDA and other such technologies.

Until now we have been looking into possible linear as well as meet in the middle attacks. But since, even though fiestal based ciphers are not susceptible to differential cryptanalysis, possible in-depth analysis of differential cryptanalysis can be done to find out if there are any weaknesses exhibited by 3DES towards differential cryptanalysis.

Chapter 6

References

[1] Cryptography and Network Security Principles and Practices, Fourth Edition by William Stallings, Fourth edition 2005.

[2] Triple Data Encryption Standard (3-DES), U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology, Oct 1999.

[3] "Data Encryption Standard (DES)", Federal Information Processing Standard Publication, FIPS PUB 46-3, National Bureau of Standards, 1977.

[4] Hamdan.O.Alanazi, B.B.Zaidan, A.A.Zaidan, M.Shabbir, “New Comparative Study Between DES, 3DES and AES”. Journal of Computing, VOL.2,No.3,pp 15-64,March 2010.

[5] M. Matsui, "Linear cryptanalysis method for DES cipher", Advances in Cryptology - EUROCRYPT'93 (Lecture notes in Computer Science No- 765), Springer- Verlag, pp. 386-397, 1994.

[6] E. Biham and A. Shamir, " Differential Cryptanalysis of DES like Cryptosystems", Journal of Cryptology, Vol. 4, no. 1, pp. 3-72,1991.

[7] K. Aoki and Y. Sasaki, " Meet in the Middle Pre-image Attacks against Reduced SHA-0 and SHA-1", Lecture notes in Computer Science, vol. 7073, pp. 344-371, Springer 2011.

[8] Shaunak S.Ganorkar, Shilpi U.Vishwakarma, Sagar D.Pande, " An Information Security Scheme for Cloud based Environment using 3DES Encryption Algorithm ", International Journal of Recent Development in Engineering and Technology, Vol.2, No.4, April 2014.

[9] William J.Buchanan, “3DES encryption in .net“, March 2011.

[10] Majithia Sachin, Dinesh Kumar, “Implementation and Analysis of AES, DES and Triple DES on GSM Network”, IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.1,pp 133-138, January 2010.

[11] S. Praveen, M. Nagesh, “Implementation of the Triple DES Block Cipher using VHDL”, International Journal of Advances in Engineering & Technology, pp. 117-128, VOL.3,No.1, 2012.

[12] P. Kitsos, S. Goudevenos, "VLSI implementations of the triple-DES block cipher”, Electronics, Circuits and Systems, ICECS 2003, 10th IEEE International Conference, pp 76-79, VOL. 1,2003.

[13] Malik Sikander Hayat Khiyal, Aihab Khan, and Khansa Shabbir, "Performance Evaluation of Encryption Techniques for Confidentiality of Very Large Databases", International Journal of Computer Theory and Engineering, Vol. 3, No. 6, December 2011.